ADAPTIVE AND DYNAMIC MESHING METHODS FOR NUMERICAL SIMULATIONS

A Thesis
Presented to
The Academic Faculty

by

Nazmiye Acikgoz

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Aerospace Engineering

Georgia Institute of Technology
May 2007

ADAPTIVE AND DYNAMIC MESHING METHODS FOR NUMERICAL SIMULATIONS

Approved by:

Professor Carlo L. Bottasso,
Committee Chair
School of Aerospace Engineering
Georgia Institute of Technology

Professor Luca Dieci
School of Mathematics
Georgia Institute of Technology

Professor Lakshmi N. Sankar
School of Aerospace Engineering
Georgia Institute of Technology

Professor Zvi Rusak
Department of Mechanical, Aerospace,
and Nuclear Engineering
Rensselaer Polytechnic Institute

Professor Stephen M. Ruffin
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: 31 January 2007

To my parents Medine & Emin Acikgoz

and

to my dear husband Sermet

ACKNOWLEDGEMENTS

Completing a PhD is truly a long journey with many ups and downs and if it was not for the great people surrounding me, I wouldn't have been able to conclude this endeavor. I must first express my gratitude toward my advisor Dr. Carlo Bottasso. His enthusiasm, hard work and brilliant mind have set an example which I wish to match someday. I am so fortunate to have had the opportunity to work with him. He was considerate and very supportive all along, I really appreciate all he has done for me both professionally and personally.

I would like to thank to my committee members Dr. Lakshmi Sankar, Dr. Stephen Ruffin, Dr. Luca Dieci and Dr. Zvi Rusak for their time, effort and enlightening suggestions which were important in improving the quality of this research. I am grateful to Dr. Jeff Jagoda for supporting me through my graduate studies and financial problems. My special thanks goto Dr. Sankar whose door was always open when I needed his valuable advises. His courteous personality encouraged me to ask his help during my hard times at Georgia Tech.

I would also like to thank to my previous lab mates who are scattered all around the world now: Julide Topsakal, Faik Sumer, Luca Riviello and Dr. Chong-Seok Chang, and my current lab mates Rene and Maxime for creating a friendly and comforting environment in the lab. I feel fortunate to have met you all. I thank to my good friend Sinem Gokgoz-Kilic, with whom I shared many coffee breaks and had many joyful conversations. I am indebted to Dr. Davide Detomi whom I have bothered with many technical questions past few years. His patiently-written detailed answers, and helpful discussions were much appreciated.

I am thankful to my very first room mate and friend Ruth Lu for helping me out during my first days in Atlanta. I felt less homesick because she was there like a good sister. I would like to express my sincere thanks to my dear aunt Zeliha Haciibrahimoglu for her endless support and encouragement. If it wasn't for her I would have never come to Atlanta for my

doctoral studies. I hope I have inherited some of her venturous and confident personality.

I am very grateful to my dear mother and father, whose many, many sacrifices in life for their children allowed me to reach this point. Their unconditional love and support made me achieve many things that otherwise would be impossible. I want to extend them my heartfelt thanks for all they have done and gone through for me. Mom and dad: here, thousands of miles away from you, I missed you very much, and every week I impatiently looked forward to our phone calls even though I knew that mom would bug me with question "When are you gonna graduate?".

My beloved husband Sermet has stood by me during all my hard times, without his guidance and constant support I would be lost. He held my hand many times I was tumbling and ready to give up. My dear: I can not thank you enough, you make everything worthwhile. Thank you for cheering me up and telling me that everything is gonna be fine when I feel hopeless. I wouldn't have come so far without you. Words can never express my love for you or my gratitude for all you bring to my life.

Finally, I want to acknowledge the financial support of Turkish Scientific and Technological Research Center (TUBITAK) during my first semester at Georgia Tech. I am also grateful to School of Aerospace Engineering for providing financial support through research assistantship for my doctoral studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

For the numerical simulation of many problems of engineering interest, it is desirable to have an automated mesh adaption tool capable of producing high quality meshes with an affordably low number of mesh points. This is important especially for problems, which are characterized by anisotropic features of the solution and require mesh clustering in the direction of high gradients. Another significant issue in meshing emerges in the area of unsteady simulations with moving boundaries or interfaces, where the motion of the boundary has to be accommodated by deforming the computational grid. Similarly, there exist problems where current mesh needs to be adapted to get more accurate solutions because either the high gradient regions are initially predicted inaccurately or they change location throughout the simulation. To solve these problems, we propose three novel procedures.

For this purpose, in the first part of this work, we present an optimization procedure for three-dimensional anisotropic tetrahedral grids based on metric-driven h-adaptation. The desired anisotropy in the grid is dictated by a metric that defines the size, shape, and orientation of the grid elements throughout the computational domain. Through the use of topological and geometrical operators, the mesh is iteratively adapted until the final mesh minimizes a given objective function. In this work, the objective function measures the distance between the metric of each simplex and a target metric, which can be either user-defined (a-priori) or the result of a-posteriori error analysis. During the adaptation process, one tries to decrease the metric-based objective function until the final mesh is compliant with the target within a given tolerance. However, in regions such as corners and complex face intersections, the compliance condition was found to be very difficult or sometimes impossible to satisfy. In order to address this issue, we propose an optimization process based on an ad-hoc application of the simulated annealing technique, which improves the likelihood of removing poor elements from the grid. Moreover, a local implementation of the simulated annealing is proposed to reduce the computational cost.

Many challenging multi-physics and multi-field problems that are unsteady in nature are characterized by moving boundaries and/or interfaces. When the boundary displacements are large, which typically occurs when implicit time marching procedures are used, degenerate elements are easily formed in the grid such that frequent remeshing is required. To deal with this problem, in the second part of this work, we propose a new r-adaptation methodology. The new technique is valid for both simplicial (e.g., triangular, tet) and non-simplicial (e.g., quadrilateral, hex) deforming grids that undergo large imposed displacements at their boundaries. A two- or three-dimensional grid is deformed using a network of linear springs composed of edge springs and a set of virtual springs. The virtual springs are constructed in such a way as to oppose element collapsing. This is accomplished by confining each vertex to its ball through springs that are attached to the vertex and its projection on the ball entities. The resulting linear problem is solved using a preconditioned conjugate gradient method. The new method is compared with the classical spring analogy technique in two- and three-dimensional examples, highlighting the performance improvements achieved by the new method.

Meshes are an important part of numerical simulations. Depending on the geometry and flow conditions, the most suitable mesh for each particular problem is different. Meshes are usually generated by either using a suitable software package or solving a PDE. In both cases, engineering intuition plays a significant role in deciding where clusterings should take place. In addition, for unsteady problems, the gradients vary for each time step, which requires frequent remeshing during simulations. Therefore, in order to minimize user intervention and prevent frequent remeshings, we conclude this work by defining a novel mesh adaptation technique that integrates metric based target mesh definitions with the ball-vertex mesh deformation method. In this new approach, the entire mesh is deformed based on either an a-priori or an a-posteriori error estimator. In other words, nodal points are repositioned upon application of a force field in order to comply with the target mesh or to get more accurate solutions. The method has been tested for two-dimensional problems of a-priori metric definitions as well as for oblique shock clusterings.

Chapter I

INTRODUCTION

1.1 Adaptive Meshing Approaches for Computational Mechanics

Computational simulations are indispensable instruments of analysis and design in many engineering disciplines because of their potential to reduce the number of costly physical experiments. One of the fundamental considerations in computer simulations is how to treat a continuous domain, such as a fluid, in a discretized manner. The general method is to discretize the spatial domain into small elements to form a volume mesh and then apply a suitable algorithm to solve governing equations at these discrete locations. Hence, the role of a mesh is substantial in a numerical simulation, and this role becomes even more important as the scale of the physical problem increases.

Depending on the nature of the simulation, the user might need to generate a completely new mesh or change an existing mesh. In the context of the latter, if the problem is dynamic, one may deform a current mesh to comply with the boundary conditions or adapt it in order to obtain a more accurate solution. In all of these cases, the key issue is the prevention of poorly-shaped elements because such elements can cause numerical difficulties such as ill-conditioned matrices that tend to slow or even preclude the convergence of iterative solvers. This is a major problem of numerical simulations, necessitating the use of reliable and robust algorithms in both branches of meshing: (a) generation, and (b) adaptation, which can further be categorized into optimization methods (h-refinement) and deformation methods (r-refinement).

For the first category, mesh optimization, the ultimate objective is to modify the initial mesh using strategies such as nodal movement, refinement, and coarsening to get a mesh that is in better agreement with a user-defined target. Classical optimization schemes refine or coarsen uniformly in all directions, producing isotropic meshes in which the length scales

of each element are essentially equal. The resulting mesh obtained with classical optimization is optimal only for fields that possess nearly equal gradients in all spatial directions. Therefore, directional features such as shocks, boundary layers, wakes, slip lines and vortices are not treated cost efficiently since the number of required elements increases rapidly with each isotropic refinement [52, 61]. Clearly, anisotropic mesh generation algorithms that can keep the number of mesh points affordably low, are needed. One way to obtain anisotropy in grids is the use of metrics that are mathematically defined quality measures of size, shape, and orientation. Metrics also provide a compact way of describing the desired features in the optimized mesh [48, 45, 46].

In this work, we present a three-dimensional, metric-driven automatic mesh optimizer initiated by Bottasso [21] for unstructured anisotropic tetrahedral meshes. The algorithm uses a Gauss-Seidel iterative scheme to remove elements that do not satisfy the compliance requirement, which, in this study, is based on a user-defined distance function in terms of current and target metrics. Removal of such elements is attempted using a number of local retriangulation tools and local smoothing. Each of these "meshing primitives" proposes a new configuration of local sub-mesh neighboring the candidate element for removal. This new configuration is evaluated to see if it improves the objective function within the prescribed accuracy. The procedures described here are designed so as to be usable in arbitrarily complex curved three-dimensional domains, which is achieved by interfacing the mesh optimization software and its data structures with a solid-modeler that provides a topological and geometric description of the computational domain.

Within this optimization approach, it has been observed that clusters of elements can get stuck in "frozen" configurations; in other words, none of the local retriangulation primitives may be capable of proposing a cost-improving configuration. This finding is especially evident at the proximity of the model boundary and when the optimizer is used as a mesh generator, taking as input an extremely coarse mesh of possibly only a few elements in size. In fact, in this last case, since very few degrees of freedom are available early in the process and most mesh entities are classified on the model boundary, only a limited number of retriangulation options exist. Clearly, if one insists on a strictly cost-improving criterion,

locking into frozen configurations can occur; thus, spots of poor elements may never be removed, regardless of the total number of Gauss-Seidel iterations performed.

Therefore, we improve the robustness of the Gauss-Seidel removal scheme by using a Simulated Annealing (SA) technique, with the goal of increasing the chances of avoiding entrapment into local minima of the objective function [6]. The SA method [1, 41, 69, 44] is a random-search technique inspired by the analogy between the way in which a metal cools and stabilizes into a minimum energy crystalline structure (the annealing process) and the search for the global minimum of a function. Basically, SA allows moves that do not immediately improve the cost function but that help escape local minima by jumping over the small hills in the solution space, the main cause of entrapment. With this technique, the acceptance criterion of the down-hill policy is modified such that new local triangulations are accepted based on a probability function that inserts randomness into the process. The only drawback of SA is that it takes a very long time to completely reach equilibrium, as in the metal-cooling process. Thus, we refine SA scheme by proposing a local version of it [2, 4]. This modified algorithm targets only those spots of elements that lag behind the others during the optimization process. This approach significantly lowers the computational cost of the scheme without sacrificing its ability to prevent entrapment.

In the second part of the thesis, we focus on dynamic mesh deformation algorithms because many challenging multi-physics and multi-field problems that are unsteady in nature are characterized by moving boundaries and/or interfaces. In all these cases, the motion of a portion of the domain boundary is known, and one wants to deform the rest of the mesh in order to accommodate these imposed displacements. For this purpose, a vertex repositioning problem must be solved in a robust and efficient way such that invalid elements are avoided even for large amplitude motions. Indeed, methods that can specifically deal with large deformation problems [34, 83] are needed. The basic idea behind all the methods used for this class of problems is to define a suitable fictitious elasticity problem over the domain.

The fictitious elasticity problem can either be continuous [83] or discrete [14]. For the former approach, partial differential equations are discretized in space, for example, by using

finite element methods. Alternatively, a lumped-parameter discrete structural model can be used. In this study, we consider the discrete case, in which the fictitious problem defines a suitable network of springs associated with the mesh. The problem then becomes how to construct the best possible network of springs that a) is inexpensive to compute; b) does not contain collapse mechanisms; and c) leads to graded and well-shaped deformed grids, even for large imposed displacements.

The most widely used and the simplest mesh deformation technique is the spring analogy method [14], in which each edge is replaced by a spring whose stiffness is inversely proportional to the edge length. While this classical method performs reasonably well in a number of cases, it fails as soon as the local grid motion is not small compared to the local mesh size. Unfortunately, in many practical cases, the necessary grid displacements are not small. To address this issue, torsional springs were added to the linear edge springs by Farhat et al. [34, 31]. The torsional springs, which are designed to prevent mesh entanglement, work well in practice but the method becomes computationally burdensome in three dimensions. Furthermore, its use for non-simplicial (e.g., quadrilateral, hexahedral) meshes first requires splitting quads into triangles or hexas into tetrahedra.

In this work, we develop a unified formulation for controlling collapse mechanisms. The formulation, which covers simplicial (e.g., triangles, tetrahedrons) and non-simplicial , structured and unstructured cases and allows large-amplitude motions, is based on the idea of complementing the linear edge springs with linear face-vertex springs in three dimensions or linear edge-vertex springs in two dimensions. These additional springs effectively constrain each vertex within the polyhedral ball that encloses it, contrasting the possible collapse mechanisms of the grid elements. Furthermore, the presence of additional springs is also beneficial in terms of mesh quality because these springs tend to keep each vertex close to the centroid of the ball, by pushing it away from its boundaries. We call this method as "ball-vertex", and it is the first method in dynamic mesh-deformation literature that can cover all different types of grid elements for large amplitude boundary movements.

Final part of the thesis concentrates on mesh adaptation algorithms using nodal repositioning. For problems with large spatial and temporal variations, adaptive methods are

indispensable because they provide enhanced accuracy in the solution as well as decreasing mesh dependency and user intervention. Adaptation is especially appreciated in flow problems in which the solution depends on both flow geometry and flow conditions. For example, for unsteady problems, features of flow and geometry might change continuously thereby requiring frequent remeshing. In addition, most flow simulations are required to resolve highly demanding physical features such as vortices, flow separation, and compression waves, which are characterized by regions of sharp gradients of flow variables embedded into regions of smooth variations. Therefore, one must take advantage of adaptation techniques in order to avoid intuitive and frequent remeshing of the entire domain.

While adaptivity can be achieved in several ways, in this work, we primarily focus on one in particular: the "r- method." R-adaptation, also known as the "moving-mesh method," repositions grid points as needed while preserving initial connectivity and the total number of nodes. Although they are not new among the adaptive methods, r-adaptation techniques are not commonly used in numerical analyzes, mainly because of the difficulty in developing general and robust mesh deformation algorithms. This major problem, however, could be overcome by the ball-vertex methodology, which has proven to be quite an efficient deformation tool [3, 5]. With a known error distribution, r-adaptation could assure precise solutions for governing equations. The clustering of nodes around high-gradient regions while coarsening them at low- gradient regions would enhance the solution considerably, as in problems that involve shocks, boundary layers, vortices, or similar features. Besides, r-methods provide significant advantages such as the relative ease of coding compared to topological alterations and ease of implementing the method into existing codes with fixed meshes. Furthermore, r-adaptation is the only way of adapting structured meshes if the connectivity of the mesh and number of nodes are supposed to remain unchanged.

In our metric-based r-adaptation algorithm, we reposition vertices using the previously explained ball-vertex methodology, and hence taking advantage of its ability for large amplitude repositionings. However, it is different from the pure ball-vertex technique in the sense that compression and elongation of the springs are enabled by a pre-defined force field instead of a pre-defined boundary movement. The force field, which we describe using

both a-priori and a-posteriori (Hessian recovery) error estimations, is composed of individual resultant force vectors, each acting on a single vertex. The vertices are compelled to move at a distance proportional to the magnitude of the corresponding force vector. The displacements of the nodal points are then calculated by establishing a force equilibrium throughout the mesh domain.

1.2 Mesh Quality

Both mesh adaptation and deformation has a strong relation to the concept of 'mesh quality'. In numerics, the generally accepted idea is that quality of the mesh is good, if the resulting solution is accurate and the cost is minimum i.e., the mesh has minimum number of nodes [36]. Hence, based on the capabilities of the available solvers, several quality measurements have been suggested such as aspect ratio, minimum and maximum angles. For example, elements are considered to be high quality if their minimum angle is much larger than 0^o and their maximum angle is much smaller than 180^o. This is because small angles produce ill-conditioned systems while large angles lead to discretization errors and inaccurate derivatives. However, recent studies showed that this well known strict rule can be relaxed for some solvers, and almost flat elements can be allowed if stretched elements are properly aligned with the solution anisotropy [42].

Clearly, the mesh considered to be the best for one solver, may not meet the requirements for another, which obviously indicates that assessing mesh quality based on the abilities of a numerical solver is a rather subjective approach. In fact, it is necessary to decouple mesh quality measurement from capabilities of solvers by introducing an objective assessment. This is not a trivial task because the mesh quality is highly influenced by the nature of the problem under investigation and the geometry as well as the boundary conditions.

When available, metric maps are reliable sources for independently judging the quality of the mesh. They can be defined analytically or constructed from geometrical features of the domain, from a-posteriori error analysis, or from other user-defined input [81]. Regardless of the origin, given a prescribed metric distribution, one can objectively and independently

determine the quality of a mesh and also decouple, in a natural way: solver, mesh, and error estimation. For those reasons, throughout this work we employed metrics to accurately evaluate mesh quality in both adaptation/optimization and deformation type meshing problems.

1.3 Contributions of the Thesis

This thesis contributes to the literature in three areas of meshing; mesh generation, mesh deformation, and mesh adaptation, which are also presented in Figure 1. Each article below corresponds to one principal chapter of our work.

- A novel approach is introduced to solve a well-known mesh optimization problem. The simulated annealing technique is implemented into a tetrahedral mesh optimizer in order to remove pathological spots that can form near the boundaries and interfaces. With this new optimization method, mesh optimizers can also be employed as mesh generators because simulated annealing acceptance criterion prevents early locking of the mesh due to the lack of degrees of freedom. Furthermore, with local applications of the technique, computational cost decreases significantly. Therefore, higher quality meshes can be achieved at a remarkably lower cost.

- A novel mesh deformation algorithm is developed for moving boundary problems such as fluid flows and fluid structure interactions. This method is referred to as "ball-vertex" and has a unified formulation for all types of meshes, including tet, hex, prism, pyramid, pent and their hybrid combinations. Several distinctive properties of the technique are that 1) it has a remarkably straightforward formulation, 2) it allows very large amplitudes of boundary motions, and 3) it has a broad range of applicability, including both structured and unstructured grids.

- A novel mesh adaptation technique that repositions nodal points upon application of a force field is developed. The force field is calculated based on both a-priori and a-posteriori (Hessian recovery) error estimations in order to obtain more accurate

numerical solutions. The adaptation method reduces user intervention and minimizes errors originating from improper initial mesh selections.

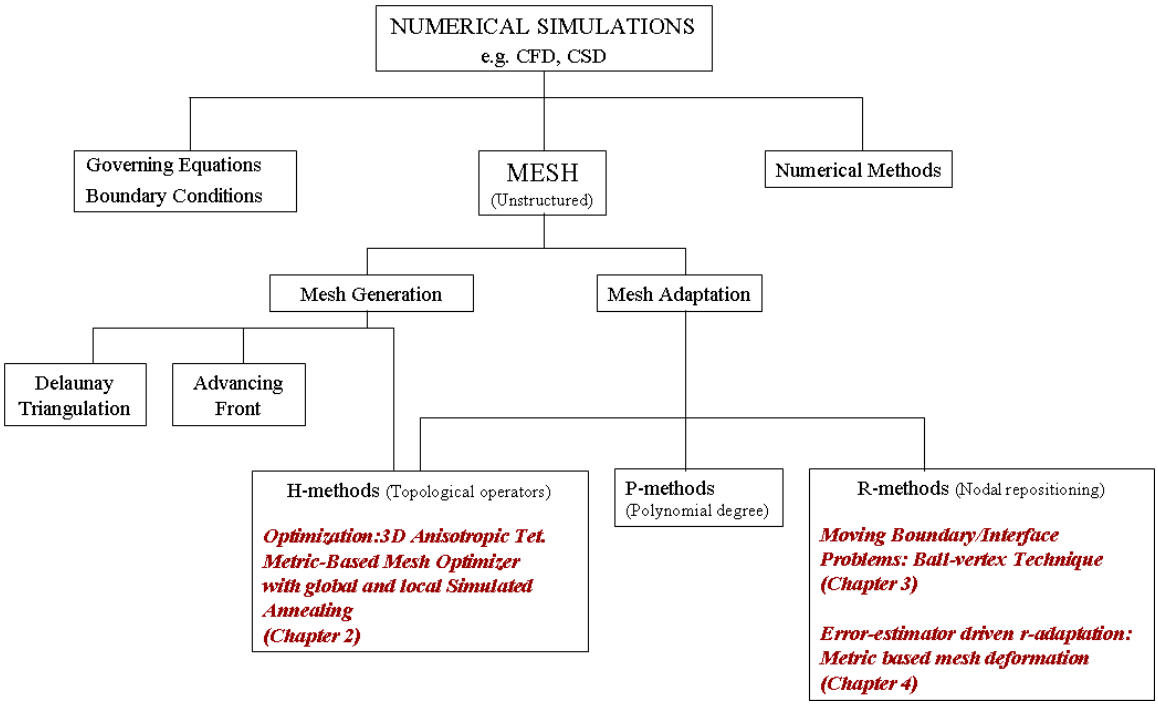


Figure 1: Contributions of the thesis

1.4 Outline of the Thesis

This thesis document is organized as follows. In Chapter 2, we explain metric driven anisotropic mesh optimization using global and local simulated annealing techniques. Section 2.3 describes the mesh optimization process and the modification primitives prior to numerical examples. Section 2.3.1 introduces map and metric definitions followed by the expressions of target metric and those of the compliance with the target. Section 2.5 presents the simulated annealing scheme and its local implementation together with several numerical examples and comparisons, and concludes with a parameter and sensitivity analysis. Furthermore Appendix A illustrates at length the interface between the geometric model and the mesh optimizer.

In Chapter 3 of the document, we present a new unified methodology for deforming both simplicial (triangular, tetrahedral) and non-simplicial (quadrilateral, hexahedral) meshes

and discuss the past studies performed in the field of mesh motion. Section 3.4 revisits the spring analogy, and Section 3.6 discusses the collapse mechanism of the current grid elements. Section 3.7 introduces the Ball-vertex formulation and controlling of the collapse mechanism with ball-vertex springs, and Section 3.8 presents applications of the ball-vertex method to several example problems, including a benchmark, an oscillating airfoil, a multi-element airfoil, a pitching-plunging, and finally, a bending wing.

In Chapter 4, we formulate a metric-based mesh adaptation methodology that employs the ball-vertex technique for moving the grid points to comply with a given target. The Chapter begins with a literature review of recent mesh adaptation studies. Section 4.4 describes the force field that deforms a net of edge and virtual springs, and Section 4.4.3 exemplifies the adaptation process in a one-dimensional problem. Section 4.5 provides numerical examples with a-priori and a-posteriori error estimators.

In Chapter 5, The major conclusions drawn from this thesis work are discussed, and Chapter 6 recommends ways to improve and expand this work.

Appendix A provides details about several tools used during the development of the above procedures. In particular, it explains the interface with a computer-aided design (CAD) system that ensures both geometric and topological validity of the local meshing operators.

This thesis contains no chapter specifically devoted to a literature survey. However the related literature reviews are presented at the beginning of the corresponding chapters.

Chapter II

METRIC-DRIVEN MESH OPTIMIZATION USING SIMULATED ANNEALING ALGORITHMS

2.1 Introduction

Unstructured grids can be obtained in several possible ways, including Delaunay insertion and mesh optimization based on local retriangulations [19, 20, 21, 23, 27, 35, 84]. These processes are typically driven by a Riemaniann metric that contains information regarding the local size and shape of the desired grid elements. The idea of metric-driven mesh generation and adaptation, first introduced in Reference [84], is now widely used. In fact, the use of metrics unifies the procedure of generating isotropic and anisotropic meshes in an elegant way. Furthermore, it decouples the problem of mesh generation/adaptation and the problem of error estimation: in this context, while the role of an error estimator is to produce a metric, the role of a mesh generation/adaptation procedure is to create a grid that satisfies this metric within a given tolerance.

As a first step in optimization, we have employed a metric-driven mesh optimization procedure using the Gauss-Seidel removal algorithm [21] in which each element in the grid is visited and an attempt is made to remove the elements that do not satisfy the goal within a prescribed accuracy by means of a number of local retriangulation primitives. The removal operators include various types of vertex insertions, edge collapsing, swaps, and vertex repositioning. The procedures described in this work interface the mesh optimization software and its data structures with a solid modeler that provides the topological and geometric description of the computational domain, as described in Appendix A.

Within this approach, we have observed that clusters of elements can get trapped in "frozen" configurations; in other words, it is possible that no one of the local retriangulation primitives available to the optimizer is capable of proposing a cost-improving configuration. This problem is especially evident when a) the proximity of the model boundary further

limits the freedom to retriangulate a cluster of elements; b) the cost function solution space is very steep [21], so that none of the possible moves results in a down-hill direction that reduces cost; and c) the optimizer is used as a mesh generator (rather than for the adaptive refinement of a grid), taking as input an extremely coarse mesh of possibly only a few elements in size. In fact, in this case, very few degrees of freedom are available early in the process, and most mesh entities are classified on the model boundary such that only a limited number of retriangulation options do exist. Clearly, if one insists on a strictly cost-improving criterion, hereby referred to as "greedy policy", locking into frozen configurations can occur, which indicates that spots of bad elements may never be removed, regardless of how many Gauss-Seidel iterations are performed.

In order to fix this bottleneck, we make use of a simulated annealing (SA) technique with the goal of improving the probability of avoiding entrapment into the local minima of the objective function [6]. The SA method [1, 41, 69, 44], previously studied in the context of mesh generation by Schumaker [72], is a random-search technique inspired by the analogy between the way in which a metal cools and stabilizes into a minimum energy crystalline structure (the annealing process) and the search for the global minimum of a function. Basically, SA allows moves that do not immediately improve the cost function but that help escape local minima by jumping over small hills in the solution space, the main cause of entrapment. Using this technique, the acceptance criterion of the down-hill greedy policy is modified such that new local triangulations are accepted based on a probability function. This effect is progressively phased out as the optimizer gets closer to the solution, using an independent variable usually interpreted as the "temperature" of the annealing process.

In this work, we study the generic SA methodology and refine it by proposing a local version of the SA scheme [2, 4]. The proposed algorithm targets only those spots of elements that lag behind the others during the optimization process, which significantly lowers the computational cost of the scheme without sacrificing its ability to avoid entrapment. The performance and robustness of the proposed methodology with respect to the algorithmic parameters is investigated with the help of a number of examples.

2.2 Literature Review

The concept of metric-driven mesh generation and adaptation, first introduced by Vallet [84] in 1992, is now widely used since it generalizes, in a natural manner, the process of generating an extensive variety of grids [21]. The metric can be defined a-priori, or it can be the result of a-posteriori error estimation. In both cases, the goal is to produce a grid made of unit equilateral simplexes in the metric space. Applications of this concept to mesh generation and adaptation are discussed in several studies, which are reviewed in the following paragraphs.

In 1997, Borouchaki et al. [19, 20] proposed a two-dimensional Delaunay-type mesh generation algorithm governed by a metric map. The algorithm inserts points with a Delaunay kernel and uses edge swapping in order to obtain desired mesh sizes that comply with an a-priori defined Riemannian metric. Another two-dimensional illustration is the solution-adapted meshes presented by Buscaglia et al. [23]. They formed a suitable metric in the computational domain by approximating the second spatial derivative of the solution, i.e., the Hessian. This a-posteriori defined metric is later used to optimize initial meshes for several analytical examples, such as Poisson and convection-diffusion problems.

Similarly, Castro-Diaz et al. [27] performed a mesh adaptation based on the a-posteriori error estimation for two-dimensional flow problems with a Delaunay-type mesh generator using a local optimization strategy. They adapted the grids after solving Navier-Stokes equations discretized with the finite element method. Then they re-ran the same flow problem with the adapted mesh using uniform initial flow conditions. Furthermore, they investigated an important issue in solution-adapted optimization processes: an accurate description of the metric. In other words, although the unstructured meshes provide the most suitable environment for the mesh adaptation, it is not always easy to define an efficient metric for the optimization problem as the metric is defined from the interpolation error of a particular flow variable, and it is not clear how this affects the system of PDE's with several flow variables. To address this problem, Felcman [35] suggested that one uses two flow properties, density and Mach number, for inviscid and viscous channel flow applications. Since then, it has been a common practice to employ either Mach number or

density or both for Hessian recovery purposes [27, 79, 38].

Numerous other researchers have attempted to more accurately express the error for mesh optimization procedures. For example, Venditti in 2002 [85] combined the Hessian recovery technique with a solution error control mechanism to increase cost efficiency. With two-dimensional laminar Navier-Stokes simulations, he proved that the new error estimation, compared to pure Hessian recovery, increased cost efficiency. Other researches conducted by Becker et al. [16], Rannacher et al. [65], Houston et al. [40], Machiels et al. [53] and Peraire et al. [59] also introduced several a-posteriori error definitions for finite element solvers. However, most are considered mathematically too complex for practical implementations.

The first example of three-dimensional mesh optimization was carried out by Tam et al. [79] in 2000, with a solution based anisotropic mesh adaptation. Their optimization process did not involve metric tensors; instead, they defined the error using an algebraic function based on flow field Hessian and optimized the mesh accordingly using the basic modification primitives for the laminar duct, the transonic circular arc and, inviscid wing-pylon-nacelle applications. Lo [51] presented another three-dimensional approach to an anisotropic refinement with the a-priori defined metrics. He simplified the mesh modification by using only edge bisection for refining, which is actually different from edge splitting, in which an edge can be split at any point along the edge. The methodology is very easy to apply, however, it does not account for the coarsening of the elements, which is a significant restraint in terms of realistic mesh generation. In other words, the meshes that this technique can optimize are limited to a few analytical example problems.

2.3 Metric-Based Mesh Optimization

The mesh optimization procedure utilizes local operators for coarsening, refining, and repositioning purposes. In order to determine where to apply these operators, a control map must prescribe the size, the shape and the orientation of the mesh elements to be built. These specifications are obtained from the metric of the transformation that maps the perfect mesh element into a unit triangle in two dimensions or a unit tetrahedron in three

dimensions.

2.3.1 Maps and Metrics

Unit equilateral simplexes in the control domain can be transformed into simplicial elements (triangles in two dimensions and tetrahedra in three dimensions) in the physical domain with a transformation map, as illustrated in Figure 2. This control map prescribes the size, the stretching, and the orientation of the elements in the physical domain. Let us denote

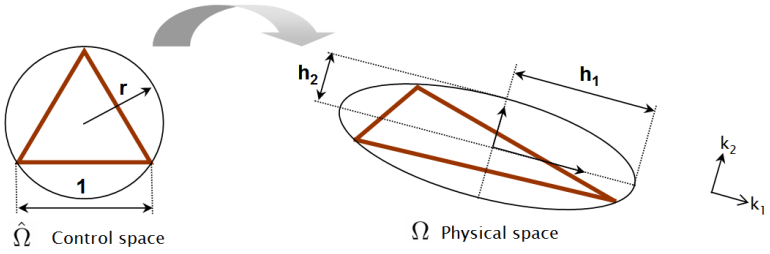


Figure 2: Mapping between the physical and the control space.

the generic simplex K, which can be obtained from the unit equilateral simplex through mapping T_K. In general, each point $\hat{P}$ with position $\hat{\boldsymbol{x}}$ in the control space (i.e., the unit simplex domain) is mapped onto point P with position vector $\boldsymbol{x}$ in the physical space [21].

$$\begin{aligned} \boldsymbol{x} &= \boldsymbol{T}_K \hat{\boldsymbol{x}} \\ &= \boldsymbol{S}_K \hat{\boldsymbol{x}} + \boldsymbol{s}_K \end{aligned} \tag{1}$$

Equation (1) shows that the transformation T_K is a combination of a translation s_K plus a rotation and deformation S_K. If we have a polar decomposition of the tensor S_K, then

$$\boldsymbol{S}_K = \boldsymbol{V}_K \boldsymbol{R}_K \tag{2}$$

which, in fact, represents the rotation and stretching of the unit simplex. Further singular value decomposition of the stretch tensor $\boldsymbol{V}_K$ gives [21]

$$\boldsymbol{V}_K = \boldsymbol{K}_K \boldsymbol{\Lambda}_K \boldsymbol{K}_K^T, \quad where \quad \boldsymbol{K}_K = [\boldsymbol{k}_1, \boldsymbol{k}_2, ..., \boldsymbol{k}_d], \quad \boldsymbol{\Lambda}_K = diag(h_{i,k}), \quad i = (1, .., d) \tag{3}$$

The $\boldsymbol{k}_i$'s form d orthogonal unit vectors that define the local principal directions of the stretch tensor, and the $h_{i,k}$'s are the local sizes in the directions $\boldsymbol{k}_i$. Furthermore, this can

geometrically be interpreted as the transformation of a unit circle into an ellipsoid with semi-axes along three orthogonal unit vectors $\boldsymbol{k}_i$, Figure 3. Lengths in the unit and actual

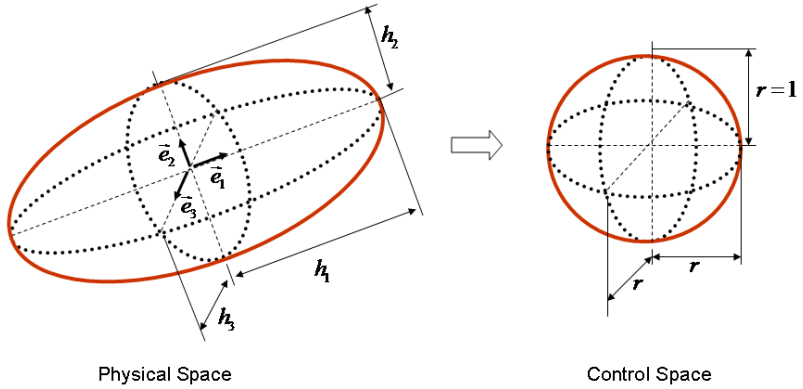


Figure 3: Geometric interpretation of Riemannian metric.

spaces are related through the metric tensor [21] by

$$\begin{aligned} ds &= \sqrt{d\hat{\boldsymbol{x}} \cdot d\hat{\boldsymbol{x}}} \\ &= \sqrt{d\boldsymbol{x}\ \boldsymbol{S}_K^{-T} \boldsymbol{S}_K^{-1}\ d\boldsymbol{x}} \\ &= \sqrt{d\boldsymbol{x}\ \boldsymbol{M}_K\ d\boldsymbol{x}} \end{aligned} \tag{4}$$

where, $\boldsymbol{M}_K$ is known as the metric tensor, and in three dimensions, it is defined at every point of the domain by a 3 x 3 symmetric positive definite matrix

$$\boldsymbol{M}_K = \boldsymbol{S}_K\ \boldsymbol{S}_K^{-1} = \boldsymbol{K}_K\, \boldsymbol{\Lambda}_K^{-2}\, \boldsymbol{K}_K^T = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \quad where\ M_{ij} = M_{ji}\ i,j = 1..,3. \tag{5}$$

Then considering Figure 3, $\boldsymbol{\Lambda_K} = \begin{pmatrix} h_1 & 0 & 0 \\ 0 & h_2 & 0 \\ 0 & 0 & h_3 \end{pmatrix}$ *and* $\boldsymbol{K}_K = [\vec{e_1}, \vec{e_2}, \vec{e_3}]$ for the ellipsoid [82]. The diagonal terms correspond to the eigenvalues of $\boldsymbol{M}_K$, which are, in fact, reflected in the width, height, and depth of the ellipsoid while the columns of the rotation matrix correspond to the eigenvectors of $\boldsymbol{M}_K$, the axes of this ellipsoid. Now one can calculate the length of edge E in the unit-simplex space $L_{M,E}$ [21] as

$$L_{M,E} = \sqrt{\boldsymbol{e}_{ij} \boldsymbol{M}_K \boldsymbol{e}_{ij}} = \sqrt{\hat{e_{ij}}\, \boldsymbol{S}_K^T \boldsymbol{S}_K^{-T} \boldsymbol{S}_K\, \hat{e_{ij}}} = 1, \forall E \in E_K, \tag{6}$$

where E is a generic edge of simplex K, which joins vertices i and j and denotes the corresponding edge vector such that $\boldsymbol{e}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$. Then $\hat{\boldsymbol{e}_{ij}}$ is the unit vector of the unit equilateral simplex, which is related to e_{ij} through $\boldsymbol{e}_{ij} = \boldsymbol{S}_K \hat{\boldsymbol{e}_{ij}}$. Therefore, the edge length of E is equal to 1 in the control space (i.e., Riemannian metric space), while it is defined in actual space (i.e., Euclidean space) as $\sqrt{\boldsymbol{e}_{ij}\,\boldsymbol{e}_{ij}}$. Later on, this property will help us calculate the elements of the metric tensor.

2.3.2 Computing the Metric Tensor $\boldsymbol{M}_K$

We will use the property in equation (6) to compute the components of the metric tensor for a given simplex K. Let's write equation (6) in the following [21] way:

$$L^2_{M,E} = \boldsymbol{e}\boldsymbol{M}_K\boldsymbol{e} = 1. \tag{7}$$

The edge vector e_{ij} is represented in short, by e here and it has the components e_1, e_2, e_3, i.e., $e = (e_1, e_2, e_3)^T$. In three dimensions, equation (7) forms a linear system of equations with metric components being unknowns, such that

$$A\boldsymbol{m} = I, \tag{8}$$

where $m = [M_{11}, M_{12}, M_{13}, M_{22}, M_{23}, M_{33}]$, and $I = [1, 1, 1, 1, 1, 1]^T$ with the ith row of matrix A written as

$$a_i = (e_1^2, 2e_1e_2, 2e_1e_3, e_2^2, 2e_2e_3, e_3^2) \quad \forall E \in E_K \tag{9}$$

In other words, we have six equations with six unknowns, which are the elements of the symmetric metric tensor.

2.3.3 Target Metric $\overline{\boldsymbol{M}_K}$

The basic idea behind metric-driven mesh optimization is to match the target metric $\overline{\boldsymbol{M}}_K$ with the actual metric tensor of the grid $\boldsymbol{M}_K$, that is, $\boldsymbol{M}_K = \overline{\boldsymbol{M}}_K$. When this equation is satisfied for each simplex, the grid is said to be "compliant to the target." In this work, we enforce this condition in an approximate manner in the mesh optimization process. Before describing the algorithm, we will explain how one can specify the target metric itself.

If the features of the final mesh are known in advance, the target metric is constructed a-priori by a user-specified input. On the other hand, when an adaptive analysis is applied, an a-posteriori error estimator can be employed to obtain the local target metrics. The latter is especially helpful in terms of adapting the mesh to follow unsteady solutions. In this case, the knowledge about the solution is converted into a form that contains information about the size and the direction.

In a typical case of an a-priori metric description, the target metric $\overline{\boldsymbol{M}(\boldsymbol{x})}_K$ can analytically be specified for each point of the domain. A function takes as input the position vector of the point, then evaluates the metric and returns the components of the metric tensor. One can easily express the desired element sizes and orientations with the help of several algebraic functions. The definitions of metric tensors in this group can be better understood through the simple examples provided below.

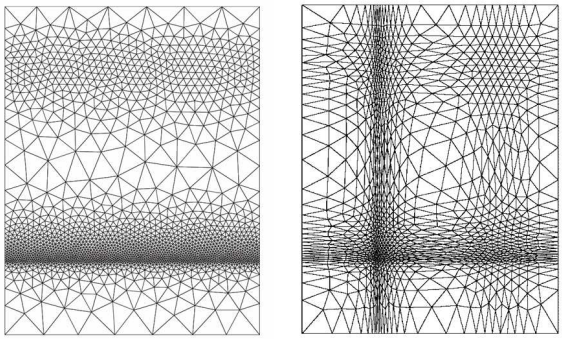


Figure 4: Mesh obtained with isotropic and anisotropic metrics [81].

This first example [81] describes an isotropic metric on a rectangular domain with ranges [0,7] x [0,9] by $\bar{\mathbf{M}}_{\mathbf{K}} = [h^{-2}\ 0;\ 0\ h^{-2}]$, where h, the desired element size, is given by

$$\mathbf{h} = \begin{cases} 1 - 19y/40 & if y \in [0, 2] \\ 20^{(2y-9)/5} & if y \in [2, 4.5] \\ 5^{(2y-9)/5} & if y \in [4.5, 7] \\ 1/5 + (y-7)^4/20 & if y \in [7, 9] \end{cases} \tag{10}$$

Figure 4 (left) provides a visual clue for the features of this metric. As can be seen, the refined mesh is clustered around $y = 2$ and $y = 7$ both horizontally and vertically.

The second test case is a special form of an anisotropic metric characterized [81] by

$\bar{\mathbf{M}}_{\mathbf{K}} = [{h_1}^{-2}\ 0;\ 0\ {h_2}^{-2}]$, where h_1 and h_2 are given by

$$\mathbf{h_1} = \begin{cases} 1 - 19x/40 & if x \in [0,2] \\ 20^{(2x-9)/5} & if x \in [2,3.5] \\ 5^{(2x-9)/5} & if x \in [3.5,5] \\ 1/5 + (x-7)^4/20 & if x \in [5,7] \end{cases} \quad \mathbf{h_2} = \begin{cases} 1 - 19y/40 & if y \in [0,2] \\ 20^{(2y-9)/5} & if y \in [2,4.5] \\ 5^{(2y-9)/5} & if y \in [4.5,7] \\ 1/5 + (y-7)^4/20 & if y \in [7,9] \end{cases} \tag{11}$$

The final refined triangulation of the domain is presented in Figure 4 (right). Note that this time the clustering is anisotropic with local mesh sizes described by h_1 and h_2. In both of these examples, the principal axes were the x and y directions, such that the rotation matrix is, in fact, the identity matrix. Later on, the results section will provide problems with more sophisticated target metrics.

The second group of metrics, i.e., a-posteriori, are in general defined in a discrete manner, through the use of an error-estimator since the problem solution is usually known at discrete locations of an initial mesh. The metric information is associated with the topological entities of the mesh such as vertices and regions, and therefore, the initial mesh must be stored unaltered as a background grid throughout the optimization while a duplicate of it is modified by the optimization or the adaptation algorithm.

One form of a-posteriori metric is obtained through the use of Hessian of the solution [84], i.e., the second derivatives of the solution field. Let us define the Hessian matrix H, evaluated at vertex P, for the approximate solution u as

$$\mathbf{H(P)} = \begin{bmatrix} u_{x_1x_1} & u_{x_1x_2} & u_{x_1x_3} \\ u_{x_2x_1} & u_{x_2x_2} & u_{x_2x_3} \\ u_{x_3x_1} & u_{x_3x_2} & u_{x_3x_3} \end{bmatrix} \ with\ \ u_{x_ix_j} = \frac{\partial^2}{\partial x_i x_j} \tag{12}$$

We then decompose the Hessian matrix into its diagonal and rotational components

$$\mathbf{H(P)} = R \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} R^{-1}. \tag{13}$$

where each $|\lambda_i|$, i.e., absolute eigenvalue of the Hessian matrix, corresponds to the target element size along the principle axes which are defined by the columns of the rotation matrix

R. Since the target metric must be a positive definite matrix, we write the tensor as

$$\mathbf{M}(\mathbf{P}) = R \begin{bmatrix} |\lambda_1| & 0 & 0 \\ 0 & |\lambda_2| & 0 \\ 0 & 0 & |\lambda_3| \end{bmatrix} R^{-1}. \tag{14}$$

Although in many adaptation problems equation (14) is used directly, for some others involving regions of uniform and/or linear solution, (i.e., $H = 0$) the target metric must be modified [35] as follows

$$M(P) = [I + Ch_{norm} R|\Lambda|R^{-1}] \cdot c, \tag{15}$$

where I is a unit 3x3 matrix, and both C and c are constants or functions, several definitions of which are given in [35], and $h_{norm} = \max_{1,j=1,..3} |\Lambda_{i,j}|$.

2.3.4 Measuring Metric Compliance

The mesh generation and/or mesh optimization process is an iterative procedure that approximates the grid compliance condition to a given tolerance by producing new mesh elements, starting with the initial simplex elements [21]. Throughout the process, both the mesh metric and the target metric will differ by a certain distance, which can be formally expressed as $d_{M_K,\bar{M}_K} = D(M_K, \bar{M}_K)$, where $d_{M_K,\bar{M}_K} \geq 0 \quad d_{M_K,\bar{M}_K} = 0 \quad iff \quad M_K = \bar{M}_K$. To measure this deviation and minimize it at the end, we introduce a non-dimensional residual function f_K given by

$$f_K = D(M_K, \bar{M}_K) D^{-1}(M_K, \bar{M}_K) \tag{16}$$

There are many different choices of the metric distance evaluations. We will explain only two of them, which were found to be the most useful and utilized in the example problems.

2.3.4.1 Metric-edge length

Measuring the distance or difference between two metrics can possibly be done by means of metric edge length (MEL). MEL involves calculating edge lengths with respect to the target metric:

$$L_{\bar{M},E} = \sqrt{e\, \bar{M}_K\, e} \tag{17}$$

Since it is already known that the length of the edge with respect to the actual metric is unity, the residual function for edge E of simplex K [21] becomes

$$f_{E,K} = (1 - L_{\bar{M},E})(1/L_{\bar{M},E} - 1) \tag{18}$$

In order to evaluate the corresponding residual for the simplex itself, we need to average $f_{E,K}$ over all the edges of the simplex such that

$$f_{K,MEL} = \sum_{E \in \{E\}_K} \frac{1}{6} f_{E,K} \tag{19}$$

where $\{E\}_K$ is the list of the edges of K.

2.3.4.2 Metric-inscribed radius

Figure 5 (left) shows a circle with radius r, which inscribes the triangle. In three dimensions, the triangle will be replaced by a tetrahedron and it circumscribe a sphere (right). The

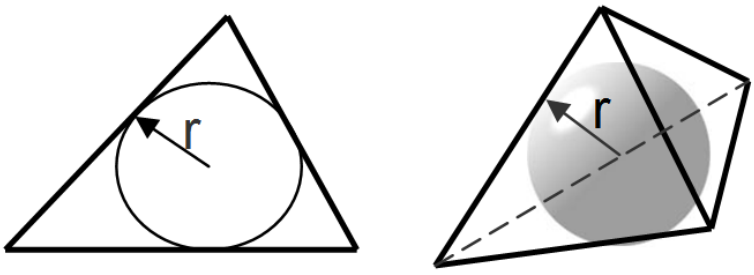


Figure 5: Description of the inscribed radius

inscribed radius of the sphere with respect to the target metric can be written as

$$r_M = \frac{3V_{\bar{M}}}{\sum_{i=1}^{4} A_{\bar{M},i}} \tag{20}$$

where $V_{\bar{M}}$ is the metric volume, and $A_{\bar{M},i}$ is the metric areas of the four tetrahedral faces. The inscribed radius with respect to the actual metric is simply the radius of the sphere, circumscribed by the equilateral tetrahedron and equal to $1/2\sqrt{6}$. Then the compliance residual based on the inscribed radius [21] becomes

$$f_{K,MIR} = (1/2\sqrt{6} - r_{\bar{M},E})(1/r_{\bar{M},E} - 2\sqrt{6}) \tag{21}$$

Neither the residual function with a metric edge length nor the residual function with an inscribed radius (MIR) alone is sufficient to measure the deviation because the former is sensitive only to the size of the simplexes and the latter only to their shape. Hence, they should be combined so that they have complete control over the mesh. For this purpose, we introduce the combined residual function

$$f_{E,MEL,MIR} = (f_{K,MEL} + f_{K,MIR})/2 \tag{22}$$

2.3.5 Gauss-Seidel Simplex Removal

With the combined definition of the residual function in equation (22), the objective function J for the optimization process over the grid $\mathcal{T}_h$ is

$$\min_{\mathcal{T}_h} J(\mathcal{T}_h) \tag{23}$$

Thus, our purpose is to minimize this objective function, which is, in fact, the maximum error over the entire grid using a Gauss-Seidel iterative method. After each simplex K is examined, its residual function is evaluated. If f_K is greater than certain tolerance ϵ, then it is inserted into a list of elements that will later undergo retriangulation by the removal operators. At the end of this removal attempt, there exists two lists of simplexes K_{old} and K_{new}. Objective functions for these sub-meshes are compared in order to determine whether the removal operation is valid [21], based on the formula

$$J(\{K_{new}\}) \leq J(\{K_{old}\}) - \delta \tag{24}$$

where δ stands for the abatement rate. If this condition is not satisfied, then this means new triangulation worsens the mesh quality and rendering it unacceptable. On the other hand, if it improves the objective function, then the grid is updated with the new list of elements. The elements that could not be eliminated with the removal process will later be operated again in a different iteration step. At the end of all iterations, such elements may remain as they were initially, or they might be removed at a later iteration because their neighboring elements have changed. Figure 6 illustrates the implementation of this algorithm. The basic component of the mesh optimization process is the simplex removal

input : $\mathcal{T}_h, \varepsilon, i_{\max}$

$\mathcal{Q} = \{K | f_K > \varepsilon \quad \forall K \in \mathcal{T}_h\}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Inizialize queues with random ordering of K in $\mathcal{T}_h$]*
do
  check = 0
  while ($K = \text{pop}(\mathcal{Q})$)
    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}(K)$ *[Try to remove bad element]*
    if ($\{K_{\text{new}}\}$) *[Removal was successful]*
      delete $\{K_{\text{old}}\}$ from $\mathcal{T}_h$, insert $\{K_{\text{new}}\}$ into $\mathcal{T}_h$ *[Update $\mathcal{T}_h$]*
      delete $\{K_{\text{old}}\}$ from $\mathcal{Q}$ and $\bar{\mathcal{Q}}$, insert $K' \in \{K_{\text{new}}\}$ into $\bar{\mathcal{Q}}$ if $f_{K'} > \varepsilon$ *[Update queues]*
      check = 1 *[Mark that a change was made]*
      end if
    else if
      push($\bar{\mathcal{Q}}, K$) *[Put bad element in work queue]*
    end if
  end while
  if ($\bar{\mathcal{Q}}$) *[Not all bad elements were removed]*
    $n_{\text{iteration}} = n_{\text{iteration}} + 1$ *[Increase iteration number]*
    $\mathcal{Q} = \bar{\mathcal{Q}}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Reinizialize queues]*
  end if
while (check **and** ($n_{\text{iteration}} \leq n_{\text{iteration,max}}$))

Figure 6: Gauss-Seidel mesh optimization algorithm.

operator. It consists of seven primitives, namely *vertex movement, edge split, edge collapse, edge swap, face swap, face split, and region split*. Each primitive operates on some entity T of the simplex K or on K itself. Names of the primitives already provide clues about which operator can act on which entity. For example, vertex movement is only applicable to the vertices of the simplex, while region split operates on the simplex K only. Later on, we will examine how each of the primitives operates in detail. First, let us begin by explaining the "best operator-best entity" strategy, employed in this study.

The strategy "best entity" refers to the process of trying all entities that the primitive is applicable to and choosing the one that most improves the objective function. Whereas, the strategy "best operator" refers to the process of trying all primitives in the operator list and then identifying the primitive with the best objective function. So, there are basically two loops as described in Figure 7. For example, consider the edge split operator. First, the

operator is performed on all six edges of the tetrahedron. If let's say, splitting edge number 4 results in the best retriangulation in terms the objective function, then this retriangulation is stored in order to compare it with the other possible retriangulations obtained through different operators. A final comparison yields the best operator to be employed on the best entity, which finally provides a new list of simplexes that is accepted, and the mesh is modified with it. In the case of no improvement, simplex removal is not possible and the algorithm continues with another simplex in the queue. Although, it is not noted in the

input : $K, \{\mathbb{M}_i\}, \delta, best_{entity}, best_{operator}$

$\{K_{\text{new}}\}_{\text{best}} = \{\emptyset\}$, $\{K_{\text{old}}\}_{\text{best}} = \{\emptyset\}$ *[Initialize lists]*
$J_{\text{best}} = \infty$
for $i = 1, n_M$ *[Loop through available meshing primitives]*
  for $T \in \{T\}_K$ *[Loop on entities of K appropriate for $\mathbb{M}_i$]*
    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}_i(T)$ *[Apply mesh modification primitive]*
    $\Delta J = J(\{K_{\text{new}}\}) - J(\{K_{\text{old}}\}) + \delta$ *[Compute objective function change]*
    if $(\Delta J \leq 0)$ *[Check if acceptable]*
      if $(J(\{K_{\text{new}}\}) < J_{\text{best}})$
        $J_{\text{best}} = J(\{K_{\text{new}}\})$
        $\{K_{\text{new}}\}_{\text{best}} = \{K_{\text{new}}\}$, $\{K_{\text{old}}\}_{\text{best}} = \{K_{\text{old}}\}$
      end if
    end if
  end for
end for
return $(\{K_{\text{new}}\}_{\text{best}}, \{K_{\text{old}}\}_{\text{best}})$

Figure 7: Simplex removal algorithm $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}(K)$

algorithm, there is a certain calling order of the primitives, which proves to be the most economical in terms of computational cost. First edge split, edge collapse, edge swap and face swap are tried, and then vertex repositioning is employed and, if still no improvement has been made, the last two operators, face split and region split, are tested.

2.3.6 Summary of Basic Mesh Modification Primitives

In this section, we briefly explain how each of the meshing primitives works. For further details, please refer to Frey [36] and Bottasso [21].

Vertex movement: In contrast to other topological operators, vertex movement, illustrated in Figure 8, is actually a geometric operator such that it does not topologically alter the mesh part on which it operates. Vertex V is repositioned based on some weighted sum of the position vectors of its edge-connected vertices V_i. The trivial expression for the position

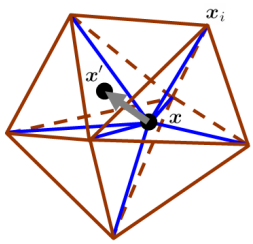


Figure 8: Vertex repositioning.

vector of V, in terms of the position vectors V_i [21] can be derived as follows:

$$\begin{aligned} \boldsymbol{x} &= \frac{1}{n_v}\sum_{i=1}^{n_v}(\boldsymbol{x}_i + (\boldsymbol{x} - \boldsymbol{x}_i)) \\ &= \frac{1}{n_v}\sum_{i=1}^{n_v}(\boldsymbol{x}_i + \boldsymbol{e}_i) \end{aligned} \tag{25}$$

where $\boldsymbol{e}_i = \boldsymbol{x} - \boldsymbol{x}_i$ indicates the edge vector that joins V to vertex V_i. The basic idea of vertex repositioning is to define a new position vector for V, $\boldsymbol{x}'$ based on new edge vectors $\tilde{e}_i$, which have unit length in the control space. Then we can write the $\boldsymbol{x}'$ and the modified edge as

$$\boldsymbol{x} = \frac{1}{n_v}\sum_{i=1}^{n_v}(\boldsymbol{x}_i + \tilde{e}_i) \tag{26}$$

$$\tilde{\boldsymbol{e}}_{\boldsymbol{i}} = \frac{\boldsymbol{e}_i}{\sqrt{\boldsymbol{e}_i \bar{M}_K \boldsymbol{e}_i}} \quad i = (1, n_v) \tag{27}$$

With the new possible position of the vertex, the relaxed position x'' is computed as

$$\boldsymbol{x''} = (1 - \omega)\boldsymbol{x} + \omega\boldsymbol{x'} \tag{28}$$

where ω, a relaxation factor, is typically chosen as 0.5 [21]. Several other vertex movement algorithms have been analyzed in the literature. The one, presented here, belongs to a class of vertex movement algorithms referred to as "variants of Laplacian smoothing". It is preferred in this case, because it has a more reasonable computation cost compared to the other methods.

Edge Split: The split location of an edge E is calculated after the opposite edge E_{opp} is determined (see Figure 9). Then the point on edge E closest to E_{opp} is chosen as the split point.

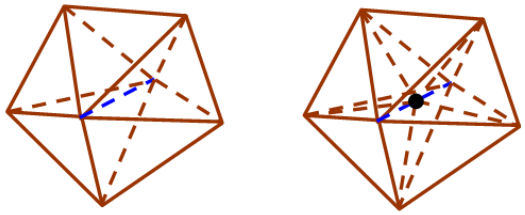


Figure 9: Splitting an edge.

Edge Collapse: Edge E has two bounding vertices V_1 and V_2, rendering two possible collapses: either V_1 onto V_2 or V_2 onto V_1 (see Figure 10). Both possibilities are investigated, and the one that is valid is chosen. In case both are acceptable, then the retriangulation with the smallest compliance residual is stored.

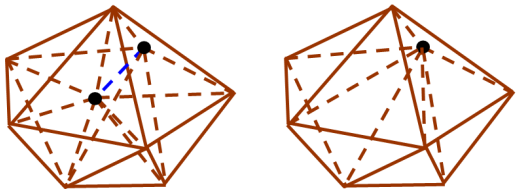


Figure 10: Collapsing an edge.

Edge Swap: In edge swap, also known as edge removal, all the tetrahedra that have "a particular edge" in common are listed as R_E (see Figure 11). The vertices of R_E minus the vertices that bound edge E form a polygon. Then the n_R-sided polygon is retriangulated into $(2n_R - 5)!/((n_R - 1)!(n_R - 2)!)$ [21] possible configurations that define $2n_R - 4$ new tetrahedra, $n_R - 2$ new faces, and $n_R - 3$ new internal edges in the cavity of the polygon. If a polygon has more than seven sides, i.e., $n_R > 7$, edge swap is not applied.

Face Swap: Face swap, also termed a "multi-face removal operator", works as follows (see Figure 12). Given a tetrahedron R_1 with a bounding face F, a vertex that is not on F, V_1 is found. Next, the tetrahedron R_2, which has the face F in common, is determined, and its vertex V_2, which is not on F, is marked. Then the face F is removed and another

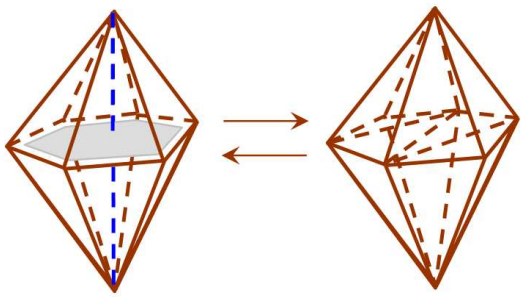


Figure 11: Swapping an edge.

face that joins V_1 and V_2 is inserted.

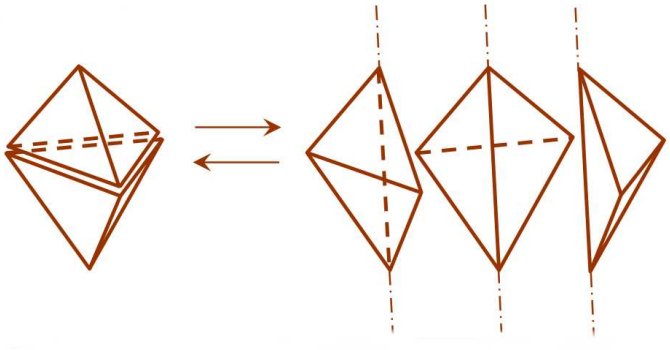


Figure 12: Swapping a face.

Face Split: Face split (see Figure 13) introduces a new vertex V on face F at some point P. Then the face is retriangulated by connecting P to the other vertices of F, as in the left part of Figure 13. The location of point P is evaluated in the same manner as the vertex movement.

Region Split: The region split operator introduces a new vertex and connects it to the 4 bounding vertices of the tetrahedron (see Figure 13, right). The position of the new vertex is computed similarly to the face split application.

2.4 Numerical Examples

2.4.1 Sphere Ridge Problem

The first problem is defined in a sphere of center C=(0.5,0.5,0.5) and radius R=0.5. The initial grid for this problem is isotropic and contains 1,678 regions and 418 vertices. Two anisotropic crests are defined, each centered at about point P_0 whose coordinates $\boldsymbol{x}_0$ are

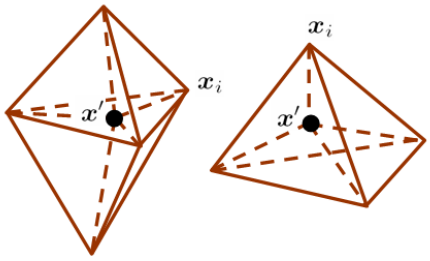


Figure 13: Splitting a face and a region.

given by

$$\boldsymbol{x}_0 = (i, i, i)^T \quad i = 0, 1 \tag{29}$$

For each generic point P in the computational domain, we define a distance d in the xy plane as

$$\boldsymbol{d} = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_0)^2 + (\boldsymbol{y} - \boldsymbol{y}_0)^2} \tag{30}$$

and a radial distance

$$\boldsymbol{r} = \sqrt{(\boldsymbol{x} - \boldsymbol{x}_0)^2 + (\boldsymbol{y} - \boldsymbol{y}_0)^2 + (\boldsymbol{z} - \boldsymbol{z}_0)^2} \tag{31}$$

$$\tilde{r} = (r - \bar{r} + a)/2a \tag{32}$$

where $\bar{r}$ indicates the distance of the crest edge from P_0, and a is the crest semi-width. For this example, $\tilde{r} = \sqrt{2}/2$, and a =0.15. Finally, the function, defining crest is expressed as

$$\mathbf{q} = \begin{cases} (1 - cos2\pi r)/2 & 0 \le r \le 1 \\ 0 & r > 1 \end{cases} \tag{33}$$

The target metric is now specified as

$$\bar{M} = \bar{D} + q(\bar{R} - \bar{D}) \tag{34}$$

where $D = diag(m_L, m_L, m_L)$ is an isotropic metric with the characteristic element size $h_L = 1/\sqrt{m_L} = 1/15$. In the region occupied by the crest, the isotropic target is replaced by the anisotropic metric [21]

$$\bar{R} = \begin{pmatrix} m_s c^2\theta c^2\psi + m_L(s^2\psi + s^2\theta c^2\psi) & (m_s - m_L)c^2\theta s\psi c\psi & (m_s - m_L)s\theta c\theta c\psi \\ & m_s c^2\theta s^2\psi + m_L(c^2\psi + s^2\theta s^2\psi) & (m_s - m_L)s\theta c\theta s\psi \\ symm. & & m_s s^2\theta + m_L c^2\theta \end{pmatrix} \tag{35}$$

where $\psi = \arctan(x - x_0, y - y_0)$ and $\theta = \arctan(z - z_0, d)$. This anisotropic target has two characteristic lengths, a smaller size in the radial direction emanating from P_0 in the yz plane, and a larger one $h_s = 1/\sqrt{m_s} = 1/100$ in the tangential direction.

Figure 14 illustrates the initial and final optimized meshes, and Figure 15 presents the histograms of the metric-edge lengths (left) and the metric-inscribed radii (right), for the initial meshes and those obtained with two different values of abatement factors. For the first histogram, we report, within each interval in the metric edge length, the number of edges whose metric length falls within the interval, normalized by the total number of edges in the mesh. For the second histogram, we give for each interval in the metric inscribed radius, the number of tetrahedra whose metric inscribed radius falls within the interval, normalized by the total number of tetrahedra in the grid. A perfect mesh should have all entities, edges for the first and regions for the second histogram, with metric values equal to 1.

The higher abatement rate has a noticeable effect on the quality of the histograms. We see that the optimization process is robust in terms of the converged value of the average compliance residuals, such that changing the optimization parameters results in almost the same final grid.

2.4.2 Wing Problem

Another numerical example is the metric-driven anisotropic mesh optimization for a wing. Several features of the flow around a wing are inherently anisotropic in nature. In fact, in general, gradients of the flow field variables are much higher in the normal direction to the leading edge than those along the span, except at the wing tip, where the flow becomes more three-dimensional and one can observe the generation of a tip vortex, again an anisotropic structure. Wakes and shock waves are additional typical solution features that become expensive to resolve with the sole use of isotropic meshes.

For an accurate anisotropic refinement of the tip vortices, the wakes and the shocks, one typically needs to conduct an adaptive analysis based on the use of an appropriate error estimator. However, important savings in grid size can still be achieved if one can

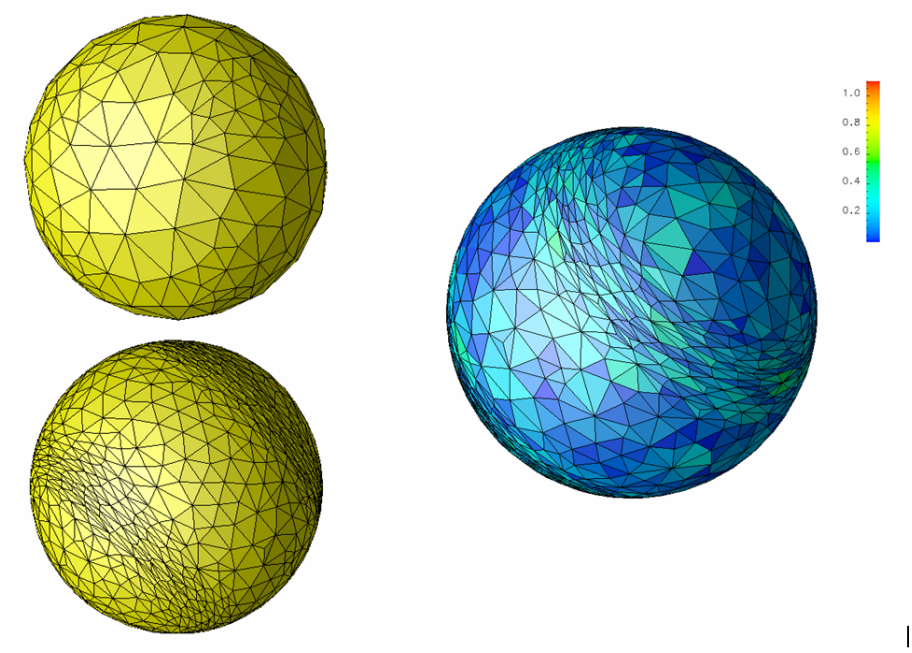


Figure 14: Sphere problem initial and optimized meshes.

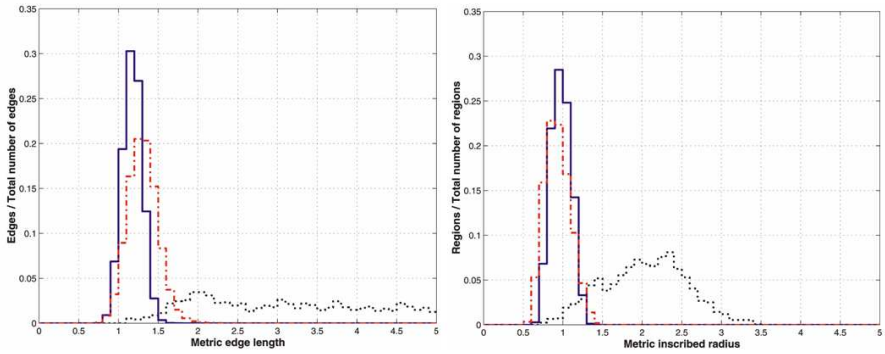


Figure 15: Left: histogram of the metric edge lengths; right: histogram of the metric inscribed radii. Dotted line: initial grid; solid line: $\delta = 0.001$; dash-dotted line: $\delta = 0.05$.

construct the initial grid with a suitable stretching along the wing span. This is especially true for high aspect ratio wings, in which the use of isotropic grids leads to unnecessarily large meshes, with a consequent waste of computing resources.

The generation of stretched initial grids is indeed possible using a metric-based optimization process. For this example, we use a wing based on a ONERA D symmetric profile with an aspect ratio of 8, a taper ratio of 0.2583, and a leading edge sweep angle of 30^o. The far field boundaries are about 50 chords from the upper and lower surfaces of the wing,

and about 15 chords from the wing tip.

The optimized mesh, shown in Figure 16, contains 219,939 tetrahedra and 39,021 vertices and has a stretching factor at the leading edge equal to 8, implying considerable savings in grid size compared to the isotropic grid. Figure 17 shows the histograms of the initial and optimized grids, which illustrate very high metric quality of the final mesh with a complete absence of poor elements.

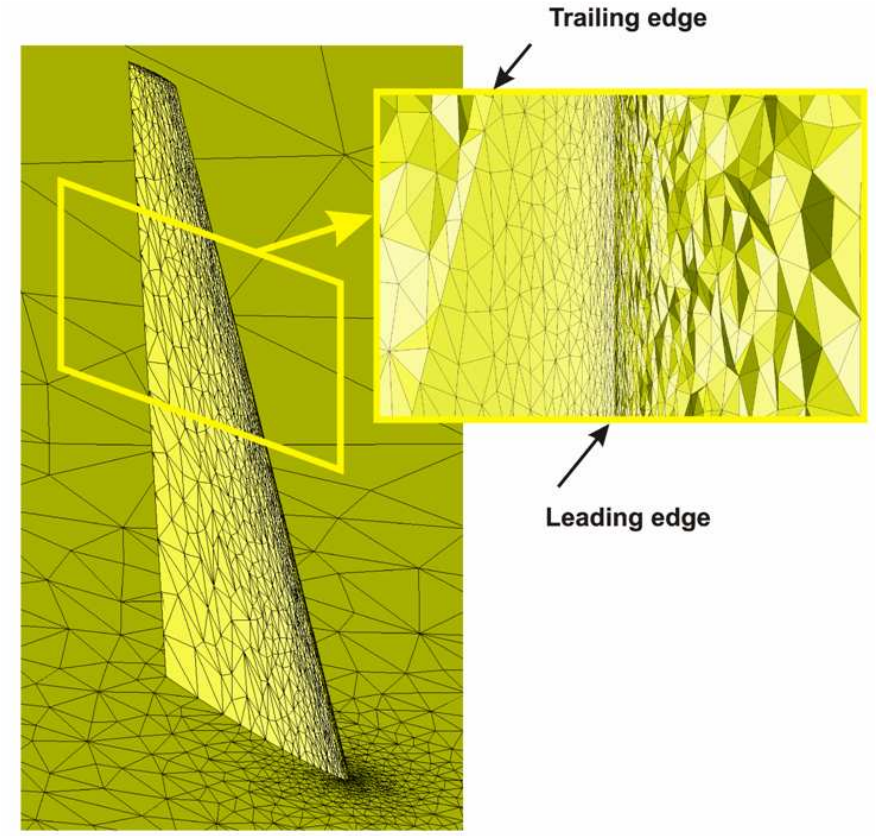


Figure 16: Anisotropic grid of a wing.

2.5 The Simulated Annealing Scheme

In many cases, the acceptance rule expressed by condition (24) works well in practice, shown in Reference [21]. However, it was observed that the procedure can become trapped in a local minimum when clusters of elements lock into a frozen configuration. These locked configurations will typically be located next to the boundary, especially in the proximity

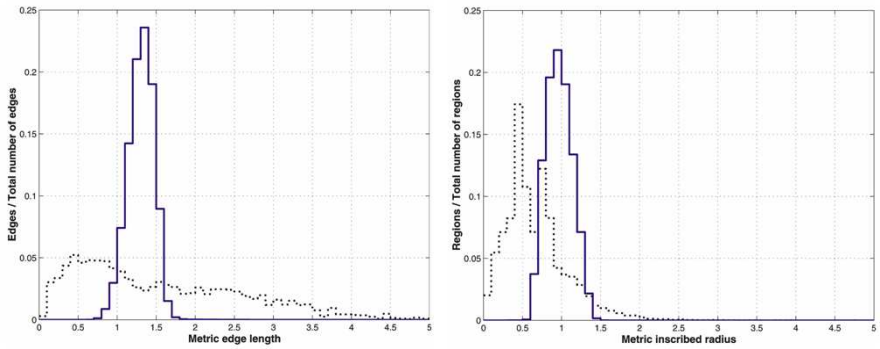


Figure 17: Left histogram of the metric edge lengths; right: histogram of the metric inscribed radii. Dotted line: initial grid; solid line: final optimized mesh.

of closely-spaced faces and multiple face intersections, where only a limited set of the retriangulations implemented in the region removal kernel $\mathbb{M}(\cdot)$ results in topologically valid moves. When this local locking occurs early in the optimization process, elements are still far from the target metric. As a result, spots of tetrahedra denoted by very poor metric quality may remain in the final grid, compromising its overall quality. More often than not, this behavior will be very localized, and one typically observes that, apart from these local problematic spots, the rest of the grid effectively converges towards the local value of the goal metric.

This behavior can be explained with the heavily constrained nature of the grid optimization process. In fact, given a point in the solution space, i.e., given the configuration of a cluster of elements, only a finite number of steps will lead to new solutions. These finite steps correspond to the finite number of possible retriangulations of the cluster. This is in contrast with other "continuous" optimization problems, in which one could move in all directions in the solution space in principle, and for each direction, one could adjust the step length at will. However, this is not possible during (topological) optimization of a grid due to the "discrete" nature of the problem. This inherent difficulty is clearly exacerbated in the proximity of the model boundary, where even less freedom is allowed.

One possible way of addressing these difficulties is to relax the strictly down-hill acceptance rule (24) with a less greedy criterion that allows for temporary mesh worsening (in the

sense of the compliance residual) during the optimization process, because allowing temporary worsening of the mesh sometimes leads to a larger set of configurations. One could have more degrees of freedom that converge to better results while exploring this wider set. One such technique is represented by the popular simulated annealing (SA) algorithm, which allows for jumps out of local minima, so entrapment into poor local configurations becomes less likely.

SA is a random search technique that follows the analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (i.e., the annealing process), and the search for a minimum in a general optimization process. In the annealing process, the material is heated to a very high temperature and then cooled slowly. While the material is hot, the annealing technique will experiment with very different configurations (since the material is malleable and its components can move around easily), but when it gets colder the configurations will only settle in those configurations that have low energy values. Once settled, the configuration will not change much since the low temperature keeps the molecules from moving around. This technique can also be used to avoid local minima that cause problems as mentioned before. Numerous applications of the technique to a variety of problems exist in the literature [41, 44, 72, 43].

Let us consider the solution space of a given optimization problem as a many-dimensional surface that consists of hills and valleys. Figure 18, illustrates a valley corresponds to a local minimum and a hill corresponding to a local maximum. The final goal of an optimization problem is to identify the global minimum or global maximum. In our case, which is a highly nonlinear problem, the global minimum of the compliance residual is being searched. Simulated annealing allows "jumps" from valley to valley during a search, giving the search algorithm the ability to escape from local optima and thus a chance to find a better solution.

Figures 19 and 20 show the new algorithms for Gauss-Seidel mesh optimization and simplex removal both incorporating SA respectively. From an algorithmic point of view, introduction of the simulated annealing into the metric-driven mesh optimization produces a relaxation of the criterion stated in equation (24). In fact, defining r as a random number such that $0 \leq r \leq 1$, being $\Delta J = J(\{K_{new}\}) \leq J(\{K_{old}\}) - \delta$ and introducing a parameter

Figure 18: Escaping from local minima with simulated annealing.

T (i.e., the so called temperature of the system), a new configuration can now be accepted if any of the following conditions are satisfied [6].

- $\Delta J \leq 0$, i.e., compliance residual computed on the new configuration is strictly smaller than the old one.
- $r \leq \exp^{-\Delta J/T}$, i.e., the new configuration is accepted with a probability that depends on the right-hand side of the previous inequality (i.e., a function of the compliance residual variation and of the temperature T of the system). Note that the exponential term is always between 0 and 1. For $\Delta J > 0$, it asymptotically tends toward 0 if temperature T decreases.

The second condition shows that the simulated annealing scheme introduces a statistical acceptance criterion. The simulated annealing process is essentially driven by the annealing schedule and the selected initial temperature, which should be high enough to "melt" the system completely and reduced towards its "freezing point." In this work, we set the initial temperature such that $\exp^{-\Delta J/T} = 0.96$ initially [66]. We also limit the total number of acceptable moves at each temperature step of the system. The idea is that if one makes too many good moves too soon, the algorithm will not find its way out of false valleys and into the one where the global minimum can be found.

Initially, when the temperature is high, the method allows high jumps from valley to valley. In fact, high values of T imply that the exponential term is near unity; hence, many low configurations (i.e., increasing the compliance residual) may easily be accepted. However, as the temperature declines, the degree of freedom reduces, and the method tends

to become identical to the strictly decreasing criterion of 24.

input : $\mathcal{T}_h, \varepsilon, n_{\text{iteration,max}}, n_{\text{accept,max}}$

$\mathcal{Q} = \{K | f_K > \varepsilon \quad \forall K \in \mathcal{T}_h\}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Inizialize queues with random ordering of K in $\mathcal{T}_h$]*
$n_{\text{iteration}} = n_{\text{temperature}} = 1$
$n_{\text{accept}} = 0$
do
  check = 0
  while $(K = \text{pop}(\mathcal{Q}))$
    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}(K)$ *[Try to remove bad element]*
    if $(\{K_{\text{new}}\})$ *[Removal was successful]*
      delete $\{K_{\text{old}}\}$ from $\mathcal{T}_h$, insert $\{K_{\text{new}}\}$ into $\mathcal{T}_h$ *[Update $\mathcal{T}_h$]*
      delete $\{K_{\text{old}}\}$ from $\mathcal{Q}$ and $\bar{\mathcal{Q}}$, insert $K' \in \{K_{\text{new}}\}$ into $\bar{\mathcal{Q}}$ if $f_{K'} > \varepsilon$ *[Update queues]*
      check = 1 *[Mark that a change was made]*
      $n_{\text{accept}} = n_{\text{accept}} + 1$
      if $(n_{\text{accept}} \geq n_{\text{accept,max}})$ *[Decrease temperature after enough removals]*
        $n_{\text{temperature}} = n_{\text{temperature}} + 1$
        $n_{\text{accept}} = 0$
      end if
    else if
      push$(\bar{\mathcal{Q}}, K)$ *[Put bad element in work queue]*
    end if
  end while
  if $(\bar{\mathcal{Q}})$ *[Not all bad elements were removed]*
    $n_{\text{iteration}} = n_{\text{iteration}} + 1$ *[Increase iteration number]*
    $\mathcal{Q} = \bar{\mathcal{Q}}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Reinizialize queues]*
  end if
while (check **and** $(n_{\text{iteration}} \leq n_{\text{iteration,max}})$)

Figure 19: The Gauss-Seidel mesh optimization algorithm with Simulated Annealing.

2.5.1 Local Simulated Annealing

The main disadvantage of the simulated annealing scheme is that computationally it takes much longer since the system must be cooled down slowly [6]. In order to solve this problem, we introduce the "local simulated annealing" technique, in which only "extremely lagging-behind" elements and their neighbors are targeted with simulated annealing. First, the elements that have relatively larger compliance residuals and the elements surrounding them are labelled as "lagging-behind," and the simulated annealing algorithm is applied to

input : $K, \{\mathbb{M}_i\}, \delta, SA, \theta(n_{\text{temperature}})$

$\{K_{\text{new}}\}_{\text{best}} = \{\emptyset\}$, $\{K_{\text{old}}\}_{\text{best}} = \{\emptyset\}$ *[Initialize lists]*
$J_{\text{best}} = \infty$
for $i = 1, n_M$ *[Loop through available meshing primitives]*
  for $T \in \{T\}_K$ *[Loop on entities of K appropriate for $\mathbb{M}_i$]*
    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}_i(T)$ *[Apply mesh modification primitive]*
    $\Delta J = J(\{K_{\text{new}}\}) - J(\{K_{\text{old}}\}) + \delta$ *[Compute objective function change]*
    if $(\Delta J \leq 0)$ *[Check if acceptable]*
      if $(J(\{K_{\text{new}}\}) < J_{\text{best}})$
        $J_{\text{best}} = J(\{K_{\text{new}}\})$
        $\{K_{\text{new}}\}_{\text{best}} = \{K_{\text{new}}\}$, $\{K_{\text{old}}\}_{\text{best}} = \{K_{\text{old}}\}$
      end if
    else if (SA **and** $J(\{K_{\text{new}}\}) < \infty$) *[If move uphill but valid, try SA]*
      $r = \text{rand(seed)}$ *[Generate random number r, $0 \leq r \leq 1$]*
      if $(r \leq e^{-\Delta J/\theta(n_{\text{temperature}})})$ *[Check if acceptable for SA]*
        if $(J(\{K_{\text{new}}\}) < J_{\text{best}})$
          $J_{\text{best}} = J(\{K_{\text{new}}\})$
          $\{K_{\text{new}}\}_{\text{best}} = \{K_{\text{new}}\}$, $\{K_{\text{old}}\}_{\text{best}} = \{K_{\text{old}}\}$
        end if
      end if
    end if
  end for
end for
return $(\{K_{\text{new}}\}_{\text{best}}, \{K_{\text{old}}\}_{\text{best}})$

Figure 20: Simplex removal algorithm $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}(K)$ with Simulated Annealing.

only those elements. The rest are governed by the standard optimization procedure. The modified algorithms with the local simulated annealing scheme are presented in Figures 21 and 22.

If the removal of element K is unsuccessful, we check to see if it is "lagging-behind" compared to the current average quality, i.e., if it is lagging behind the others in terms of residual function convergence. A "lagging-behind" element is identified by the condition

$$f_K > f_{\text{avrg}} + \alpha(f_{\max} - f_{\text{avrg}}), \tag{36}$$

where f_{avrg} and $f_{\max}$ denote the current average and worst residual function values for the whole mesh, respectively, while α is a user-defined parameter, $0 \leq \alpha \leq 1$.

Next, K and its neighboring elements are marked with a flag, which defines a local cluster of elements on which SA will be attempted. The list containing K and its neighbors, indicated as $\{\mathrm{N}_K\}$, is computed by the algorithm in Figure 21 as

$$\{\mathrm{N}_K\} = \mathrm{neigh}(K, n_{\mathrm{layers}}). \tag{37}$$

The operator $\mathrm{neigh}(\cdot,\cdot)$ finds the neighbors of K by layers. For $n_{\mathrm{layers}} = 1$, it forms a first cluster by finding all the regions that share a vertex with K; for $n_{\mathrm{layers}} = 2$, after computing the first layer cluster, it forms a second larger cluster by finding all the regions that share a vertex with the external surface of the first cluster; the operator performs similarly for higher values of the parameter n_{layers}. By increasing the n_{layers}, one increases the size of the local group of elements that, being extremely lagging-behind or close to extremely lagging-behind, will be targeted by the SA algorithm. A larger group of elements has a larger number of possible retriangulations, and hence, increases the likelihood that the pathological spot will be removed.

Once identified, the elements in $\{\mathrm{N}_K\}$ are marked with a flag so that they are distinguishable later on by the simplex removal algorithm; this marking is indicated in Figure 21 as `lagging-behind`.

$$\mathrm{mark}(K' \in \{N_K\}, \texttt{lagging-behind}), \tag{38}$$

For the rest, the local simulated annealing (LSA) Gauss-Seidel algorithm of Figure 21 is identical to the SA algorithm of Figure 19.

The simplex removal algorithm for LSA is described in Figure 22. It is nearly identical to that of Figure 20, except for the check on the applicability of SA. SA is attempted when the move is uphill ($\Delta J > 0$) and valid ($J(\{K_{\mathrm{new}}\}) < \infty$), as before, but in this case, only if the element being considered for removal belongs to a cluster of previously marked elements. This condition is detected in the algorithm of Figure 22 by the statement

$$\mathrm{flag}(K, \texttt{lagging-behind}), \tag{39}$$

where the operator $\mathrm{flag}(T, \texttt{Label})$ returns `true` if entity T is marked with `Label`.

In practice, using this algorithm, SA will target only those elements that are *lagging behind the others* in terms of current quality as measured by condition (36), since these

are probably "stuck" in some locked configuration. Although still possibly very far from the target, all other elements that behave as the average of the grid, are governed by the standard greedy strictly improving policy. In fact, in these regions of the mesh, the optimization converges nicely, and the uphill moves of SA are not needed since they could only slow down convergence.

2.5.2 Numerical Results and Comparison with Standard Optimization

Some interesting results have been achieved by using simulated annealing inside the mesh optimization process. Figure 23 represents an isotropic problem, defined in a box of sides with length 0.1 along x, 1 along y and 1 along z. The center of the box is located at point C(0.05, 0.5, 0.5). At first, a uniform grid of 24 elements is generated and then adapted in order to comply to a pre-specified target metric. At the left of the figure, the original grid, which is extremely coarse with 24 tetrahedra, and at the right, two optimized grids with (SA) and without (greedy) simulated annealing are shown. The combined compliance residual $f_{E,MEL,MIR} = (f_{K,MEL} + f_{K,MIR})/2$ was used with a tolerance $\epsilon = 0.15$ and an abatement factor of $\delta = 0.01$.

In the 'greedy' case, one can clearly observe near the boundaries some mesh areas in which the objective function still has relatively high values. This is not surprising because the model boundaries represent a constraint for the mesh entities that are classified on them; hence, practically reducing the number of allowable mesh modifications that can be applied in order to reach the desired metric. However, these worse regions disappear for solutions computed with simulated annealing.

Figure 24 exhibits the mesh quality histograms of the initial and final optimized grids. As can be seen from the plots, the initial grid is quite far from the optimized grids, which indicates that the initial grid is very poorly shaped and sized compared to the target. If we consider the optimized meshes, they do not differ significantly in terms of metric qualities; that is unacceptable elements resulting from the greedy algorithm do not show up in the histograms because they are very small in number compared to the total number of elements, which is 18,680 for standard optimized mesh.

input : $\mathcal{T}_h, \varepsilon, n_{\text{iteration,max}}, n_{\text{accept,max}}, n_{\text{layers}}, \alpha$

$f_{\max} = \max_{K \in \mathcal{T}_h} f_K, f_{avrg} = \frac{1}{N} \sum_{K \in \mathcal{T}_h} f_K$ *[Compute worst and average value of cost function]*

$\mathcal{Q} = \{K | f_K > \varepsilon \, \forall K \in \mathcal{T}_h\}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Inizialize queues with random ordering of K in $\mathcal{T}_h$]*

$n_{\text{iteration}} = n_{\text{temperature}} = 1$

$n_{\text{accept}} = 0$

do

  check $= 0$

  while $(K = \text{pop}(\mathcal{Q}))$

    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}, SA_{\text{accept}}) = \mathbb{M}(K)$ *[Try to remove bad element]*

    if $(\{K_{\text{new}}\})$ *[Removal was successful]*

      delete $\{K_{\text{old}}\}$ from $\mathcal{T}_h$, insert $\{K_{\text{new}}\}$ into $\mathcal{T}_h$ *[Update $\mathcal{T}_h$]*

      delete $\{K_{\text{old}}\}$ from $\mathcal{Q}$ and $\bar{\mathcal{Q}}$, insert $K' \in \{K_{\text{new}}\}$ into $\bar{\mathcal{Q}}$ if $f_{K'} > \varepsilon$ *[Update queues]*

      update $f_{\max}$ and f_{avrg}

      check $= 1$ *[Mark that a change was made]*

      if (SA_{accept}) *[Removal was allowed by SA criterion]*

        $n_{\text{accept}} = n_{\text{accept}} + 1$

      end if

      if $(n_{\text{accept}} \geq n_{\text{accept,max}})$ *[Decrease temperature after enough removals]*

        $n_{\text{temperature}} = n_{\text{temperature}} + 1$

        $n_{\text{accept}} = 0$

      end if

    else

      push$(\bar{\mathcal{Q}}, K)$ *[Put bad element in work queue]*

      if $(f_K > f_{\text{avrg}} + \alpha(f_{\max} - f_{\text{avrg}}))$ *[Check if "lagging-behind"]*

        $\{\text{N}_K\} = \text{neigh}(K, n_{\text{layers}})$ *[Construct a list of neighbors of K]*

        mark$(K' \in \{N_K\}, \texttt{Bad})$ *[Mark all neighbors of "lagging-behind" element]*

      end if

    end if

  end while

  if $(\bar{\mathcal{Q}})$ *[Not all bad elements were removed]*

    $n_{\text{iteration}} = n_{\text{iteration}} + 1$ *[Increase iteration number]*

    $\mathcal{Q} = \bar{\mathcal{Q}}, \bar{\mathcal{Q}} = \{\emptyset\}$ *[Reinizialize queues]*

  end if

while (check **and** $(n_{\text{iteration}} \leq n_{\text{iteration,max}})$)

Figure 21: Gauss-Seidel mesh optimization algorithm with LSA.

input : $K, \{\mathbb{M}_i\}, \delta, SA, n_{\text{temperature}}, \text{best}_{\text{entity}}, \text{best}_{\text{operator}}$

$\{K_{\text{new}}\}_{\text{best}} = \{\emptyset\},\ \{K_{\text{old}}\}_{\text{best}} = \{\emptyset\}$ *[Initialize lists]*
$J_{\text{best}} = \infty$
$\text{SA}_{\text{accept}} = 0$
for $i = 1, n_M$ *[Loop through available meshing primitives]*
  for $T \in \{T\}_K$ *[Loop on entities of K appropriate for $\mathbb{M}_i$]*
    $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}_i(T)$ *[Apply mesh modification primitive]*
    $\Delta J = J(\{K_{\text{new}}\}) - J(\{K_{\text{old}}\}) + \delta$ *[Compute objective function change]*
    if $(\Delta J \leq 0)$ *[Check if acceptable]*
      if $(J(\{K_{\text{new}}\}) < J_{\text{best}})$
        $J_{\text{best}} = J(\{K_{\text{new}}\})$
        $\{K_{\text{new}}\}_{\text{best}} = \{K_{\text{new}}\},\ \{K_{\text{old}}\}_{\text{best}} = \{K_{\text{old}}\}$
      end if
      if (**not** $\text{best}_{\text{entity}}$)
        break *[Leave loop on entities]*
      end if
[If $\Delta J > 0$ but move is valid, try SA on lagging-behind element or one of its neighbors]
    else if (SA **and** $flag(K, \text{“lagging-behind''})$ **and** $J(\{K_{\text{new}}\}) \leq \infty$)
      $r = \text{rand(seed)}$ *[Generate random number r, $0 \leq r \leq 1$]*
      if $(r \leq e^{-\Delta J / T(n_{\text{temperature}})})$ *[Check if acceptable for simulated annealing]*
        $\{K_{\text{new}}\}_{\text{best}} = \{K_{\text{new}}\},\ \{K_{\text{old}}\}_{\text{best}} = \{K_{\text{old}}\}$
        $\text{SA}_{\text{accept}} = 1$ *[Let caller know that removal was allowed by SA criterion]*
        $\text{best}_{\text{operator}} = 0$ *[force exit from loop on meshing primitives]*
        break *[leave loop on entities]*
      end if
    end if
  end for
  if ($\{K_{\text{new}}\}_{\text{best}}$ **and** (**not** $\text{best}_{\text{operator}}$))
    break *[Leave loop on meshing primitives]*
  end if
end for
return $(\{K_{\text{new}}\}_{\text{best}}, \{K_{\text{old}}\}_{\text{best}}, \text{SA}_{\text{accept}})$

Figure 22: Simplex removal algorithm $(\{K_{\text{new}}\}, \{K_{\text{old}}\}) = \mathbb{M}(K)$ with Local Simulated Annealing.

Note that in Table 1, the number of elements that have a metric edge-length greater than 2.0 is only 102 (0.5%) for the final mesh optimized with the greedy algorithm, which are not recognizable in the histogram diagrams. However, Figure 23 has easily captured them visually.

A similar simulation is shown in Figure 25 for an anisotropic case in which horizontally

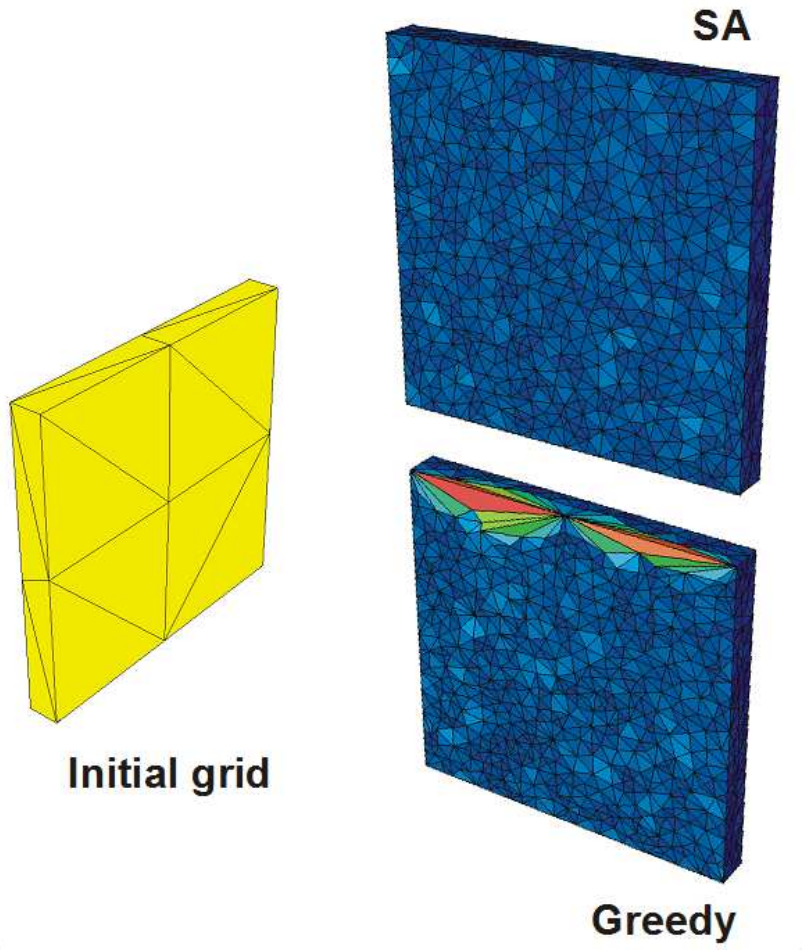


Figure 23: Isotropic problem; comparison of greedy and simulated annealing algorithms.

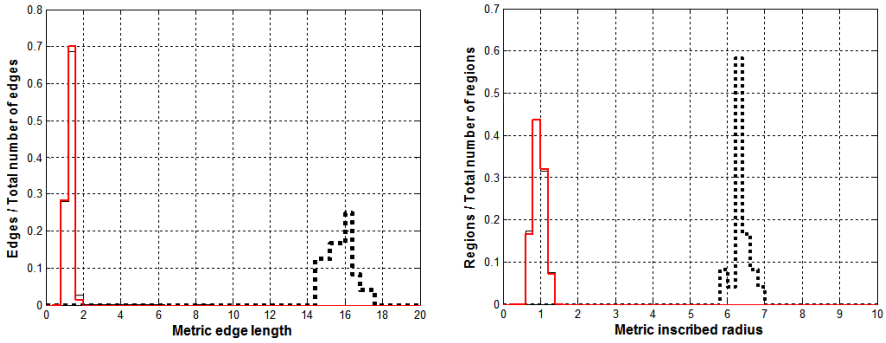


Figure 24: Metric quality histograms of metric edge length and metric inscribed radius. Red solid line : optimized mesh with simulated annealing scheme, Black solid line : optimized mesh with greedy, Dotted Black line: initial mesh.

stretched elements have been requested through the target metric and the domain has again been triangulated with both greedy (bottom) and SA (top) schemes, leading to the same

Table 1: Comparison of greedy and SA solutions for the isotropic grid problem.

	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	$N_{MEL>2}$
Initial grid	10.11	9.17	24	24
Greedy	3.30	0.09	18,680	102
SA	0.19	0.08	18,805	0

Table 2: Comparison of greedy and SA solutions for the anisotropic smooth ridge problem.

	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	$N_{MEL>2}$
Initial grid	24.81	20.23	24	24
Greedy	12.09	0.28	7,098	68
SA	0.18	0.05	9,706	0

conclusions (Table 2) previously discussed.

Another anisotropic problem is represented in Figure 26, in which the target metric defines four intersecting curved ridges. Before discussing the results, it might be interesting to examine the target metric, which creates four anisotropic curved 'crests' that intersect at the center of the box. Each crest of the anisotropic elements is centered around a point P_o, whose location in the physical domain is given by

$$\boldsymbol{x}_0 = (x_0, y_0, z_0)^T = (0, i, j)^T \quad i, j = 0, 1 \tag{40}$$

For each generic point P of coordinates $\boldsymbol{x} = (x, y, z)^T$ in physical space, we define a distance r from point P to P_0 in the yz plane as

$$\boldsymbol{r} = \sqrt{(\boldsymbol{y} - \boldsymbol{y}_0)^2 + (\boldsymbol{z} - \boldsymbol{z}_0)^2} \tag{41}$$

Next, we let

$$\tilde{r} = (r - \bar{r} + a)/2a \tag{42}$$

where $\bar{r}$ indicates the distance of the crest edge from P_0, and where a is the crest semi-width. For this example, $\tilde{r} = \sqrt{2}/2$ and a =0.1. Finally, the function defining the crest is expressed as

$$\mathbf{q} = \begin{cases} (1 - cos2\pi r)/2 & 0 \le r \le 1 \\ 0 & r > 1 \end{cases} \tag{43}$$

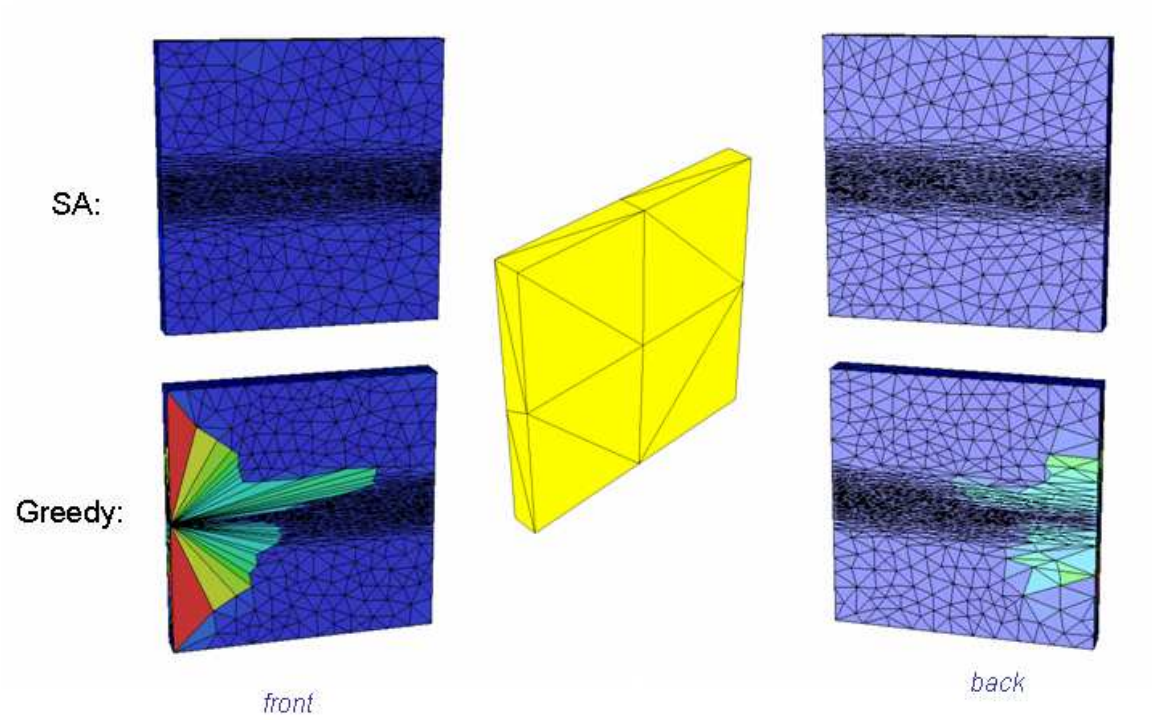


Figure 25: Anisotropic smooth ridge problem; comparison of greedy and simulated annealing algorithms.

The target metric is now specified as

$$\bar{M} = \bar{D} + q(\bar{R} - \bar{D}) \tag{44}$$

where $D = diag(m_L, m_L, m_L)$ is an isotropic metric with $h_L = 1/\sqrt{m_L} = 1/20$ characteristic element size. In the region occupied by the crest, the isotropic target is replaced by the anisotropic metric [21]

$$\bar{R} = \begin{pmatrix} m_s c^2\theta c^2\psi + m_L(s^2\psi + s^2\theta c^2\psi) & (m_s - m_L)c^2\theta s\psi c\psi & (m_s - m_L)s\theta c\theta c\psi \\ & m_s c^2\theta s^2\psi + m_L(c^2\psi + s^2\theta s^2\psi) & (m_s - m_L)s\theta c\theta s\psi \\ symm. & & m_s s^2\theta + m_L c^2\theta \end{pmatrix} \tag{45}$$

where $\psi = \arctan(y - y_0, 0)$, and $\theta = \arctan(z - z_0, abs(y - y_o))$. This anisotropic target has two characteristic lengths, a smaller size $h_s = 1/\sqrt{m_s} = 1/283$ in the radial direction, emanating from P_0 in the yz plane, and a larger size h_L in the tangential direction.

The initial and two adapted meshes are shown in Figure 26. The final mesh has 24,435 elements and 5,703 vertices for the simulated annealing solution with a tolerance $\epsilon = 0.15$ and an abatement factor of $\delta = 0.01$. This time initial mesh has 95 elements, which is still

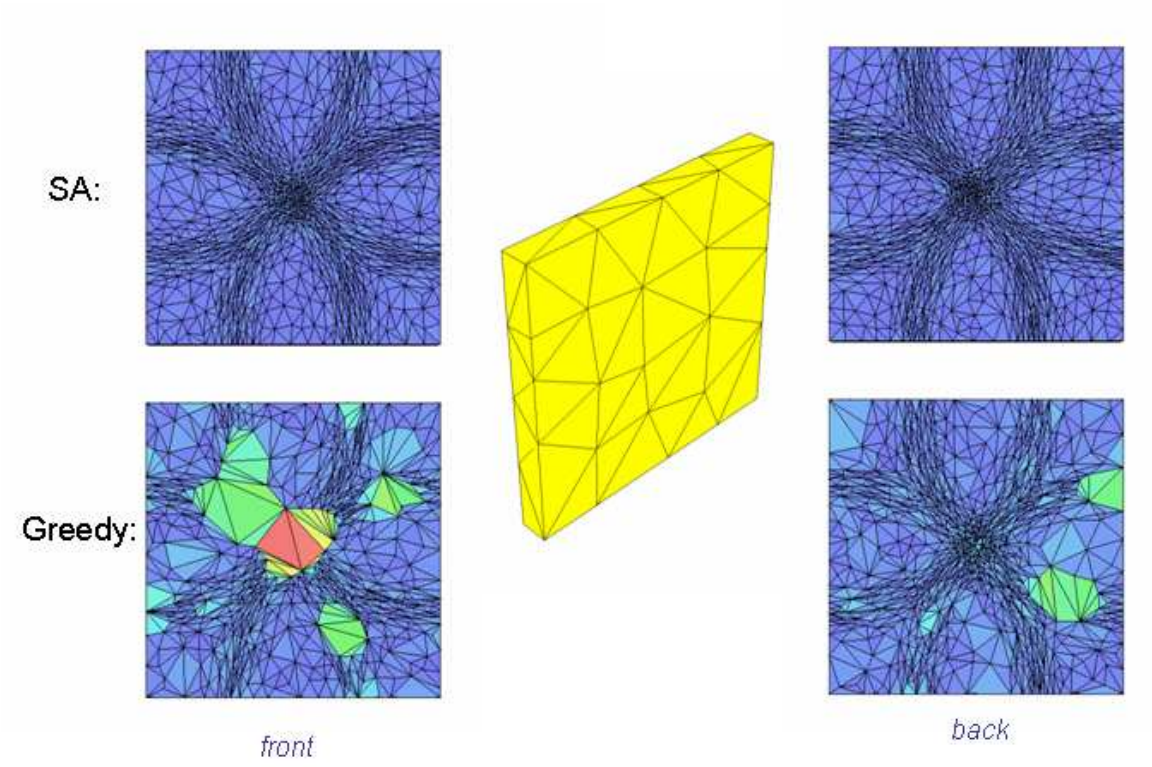


Figure 26: Anisotropic curved ridge problem; comparison of greedy and simulated annealing algorithms.

very crude. The final mesh with simulated annealing captures the features of the desired target metric well, while the greedy algorithm cannot eliminate some of the large elements on the crests. Therefore, its final mesh configuration is certainly unacceptable in terms of the target.

As shown in the three examples, it is sometimes not possible to reach a global optimum with the standard optimization scheme due to insufficient degrees of freedom. Starting from only 95 elements, which are very far from being compliant with the target, the greedy algorithm also gets stuck. At this point, we need a scheme such as simulated annealing to take over the problem. In fact, SA gives nice and smooth crests as targeted for the curved ridge problem.

Our last example problem is the application of the methodology to the generation of an anisotropic grid for a wing. For this example, we use the same wing as described in section 2.4.2. While Figure 27 exhibits the final triangulations with the two algorithms,

Figure 28 shows the initial and final triangulations from a closer view. As can be seen from the figures, at the leading edge where a high level of anisotropy is required by numerical simulation, initial poor triangulation cannot be removed using the standard technique. The greedy algorithm generates the same nice quality mesh as SA almost everywhere except near the leading edge, where in fact the high quality mesh is needed the most. Once again the statistical histograms cannot capture the discrepancy of the classical approach, shown in Figure 29. In Section 2.4.2, a nicely shaped and sized mesh has been generated even with the greedy algorithm, which was possible only because the initial mesh was relatively close to the target. In such cases, in which the target and initial metrics are too distant from each other, creating a mesh that is compliant with the given target may not be possible.

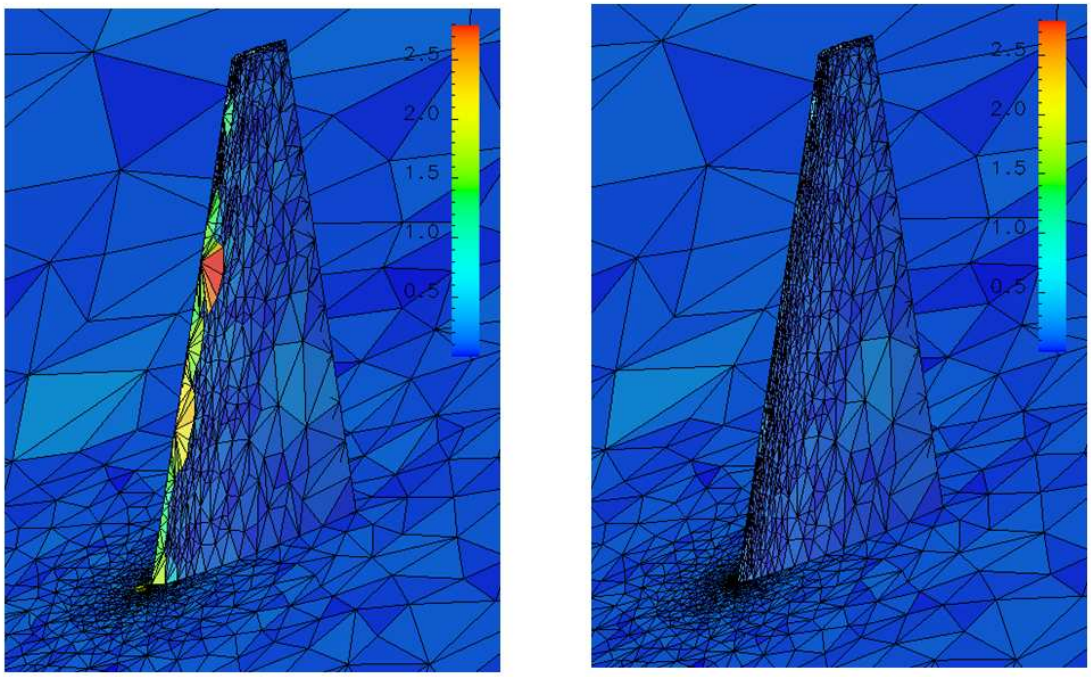


Figure 27: Gauge function distribution for wing solutions with greedy algorithm on the left and simulated annealing on the right.

2.5.3 Comparison of SA and LSA Algorithms

In the previous section, we showed, with the help of several examples, that the SA scheme is very effective at avoiding entrapment into local minima. In this section, we verify whether its local version, LSA, can produce the same quality results at a reduced computational

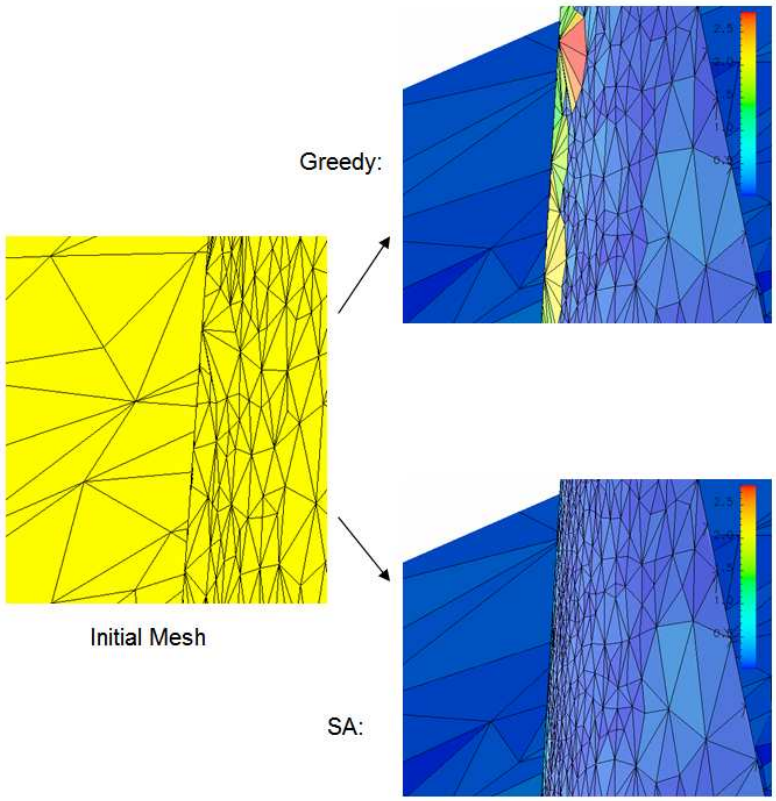


Figure 28: Gauge function distribution for wing solutions with simulated annealing and greedy algorithms: Closer look

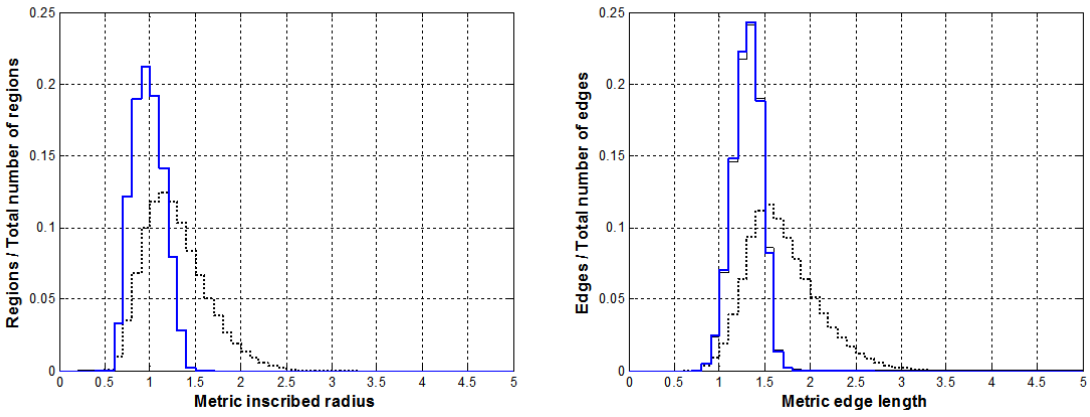


Figure 29: Metric quality histograms for the wing. Solid blue line: Simulated annealing solution. Solid black line: Greedy algorithm solution. Dotted black line: Initial mesh.

cost.

Of the extended set of tests that we have conducted, two results obtained with the previously described isotropic grid in a parallelepiped and the anisotropic wing problem are

reported here. These results are summarized in Tables 3 and 4. The normalized CPU times show that the computation time for LSA is about the same as that of the greedy algorithm and substantially less than SA. Furthermore, LSA has had to sacrifice very little in quality and this is still far superior to the greedy optimization results.

Clearly, the amount of savings in CPU usage for LSA over SA depends on the total number of elements that lag behind the others, consequently marked as "lagging-behind." For problems in which very few lagging-behind spots exist, it is quite possible to remove them very quickly. For example, in the case of the isotropic grid problem, the greedy optimized mesh has as few as 102 tetrahedra that have a residual function value greater than 2, as shown in Table 1. This number represents a rather small percentage of the total mesh size of 18,680 tetrahedra. Therefore, the time savings for LSA are significant, and the CPU usage is as small as (1.01 times) that for the greedy policy, but without its remaining lagging-behind spots. For the wing problem, more lagging-behind elements need to be eliminated, so in this case the computational effort is slightly larger, 1.08 times that of the greedy policy. Still the gain, which is accompanied by a grid quality essentially identical to the one with SA, is substantial.

The tuning parameters α and the n_{layers} of the LSA mesh optimization algorithm (Figure 21) affect the coverage area of LSA. In particular, α determines when one starts considering an element as extremely lagging-behind, i.e., as lagging behind the others in terms of convergence towards the goal, while n_{layers} creates a buffer of elements around the lagging-behind ones so that it increases the number of degrees of freedom, and hence, increases the chances of finding valid retriangulations. Therefore, we can enlarge the application area of SA simply by decreasing α and/or increasing n_{layers}. Apparently, this creates a trade-off between quality and cost. For the examples in this section, α was set at 0.95, while n_{layers} was set at 1.

2.5.4 Sensitivity to the Algorithmic Parameters

The most significant parameters for the algorithms of Figures 19, 20 and 21, 22 are the initial temperature, its schedule, and $n_{\text{accept,max}}$, i.e., the number of successful removals

Table 3: Comparison of the greedy, SA, and LSA solutions for the isotropic grid problem.

	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	$\text{CPU}/\text{CPU}_{\text{greedy}}$
Greedy	3.30	0.09	18,680	1.00
SA	0.19	0.08	18,805	1.43
LSA	0.33	0.08	18,934	1.01

Table 4: Comparison of the greedy, SA, and LSA solutions for the wing problem.

	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	$\text{CPU}/\text{CPU}_{\text{greedy}}$
Greedy	5.23	0.08	220,446	1.00
SA	0.36	0.08	221,821	1.42
LSA	0.36	0.08	221,817	1.08

required before advancing to the next lower temperature step. The purpose of this section is to show the sensitivity of the final triangulations to the variations in these parameters.

In order to study the effect of annealing temperature, Boltzman quenching and fast annealing schedules [69] are tested without observing noticeable differences in practice. All the results below use the Boltzman schedule for comparison.

Next, we investigate the influence of the initial temperature, θ_{init}. As discussed before, the essential idea behind SA is the physical analogy between the cooling of a metal and optimizing for a global minimum. From a physical point of view, the higher the initial temperature of the metal, the higher the probability of reaching the minimum energy crystalline structure, since molecules can move around more easily when they are hot. Results for varying initial temperatures are summarized in Table 5 for the isotropic box problem. For this particular case, large changes in the initial temperature do not seem to have a very noticeable effect. Apparently, "hills" in the solution space are small enough to allow any initial temperature to result in almost the same final grid. However, in the case of the smooth ridge problem, if θ_{init} is too small, the optimization ends up with clearly identifiable lagging-behind spots, as shown in Figure 30 and more quantitatively in Table 6. Examining the effect of the variations of θ_{init} for the anisotropic wing problem, reported in Table 7 leads to similar conclusions. In practice, it appears that the procedures are fairly insensitive, and

hence robust, with respect to θ_{init}, except for extremely low annealing temperatures, which are in fact approximately equivalent to a strictly improving acceptance policy.

Next, we consider the parameter $n_{\text{accept,max}}$, whose role is to avoid freezing the triangulations by making all the good moves at the very beginning of the process. In fact, if the parameter is kept too small, the procedure might run out of possibilities too early, while very large values would give rise to high CPU consumption. Table 8 shows the effects of $n_{\text{accept,max}}$ on isotropic box optimization and Table 9 for the anisotropic wing problem. For the box problem, one might notice that there is no preferable range of $n_{\text{accept,max}}$ values in terms of final quality, while the interval $1 - 20$ is associated with a lower computational cost. In the case of the wing problem on the other hand, the range $5 - 300$ provides good results both in terms of quality and cost. Comparing desirable ranges with final mesh sizes for this and other similar numerical experiments shows that a reasonable choice for this algorithmic parameter is $n_{\text{accept,max}}$=*final mesh size*/1000.

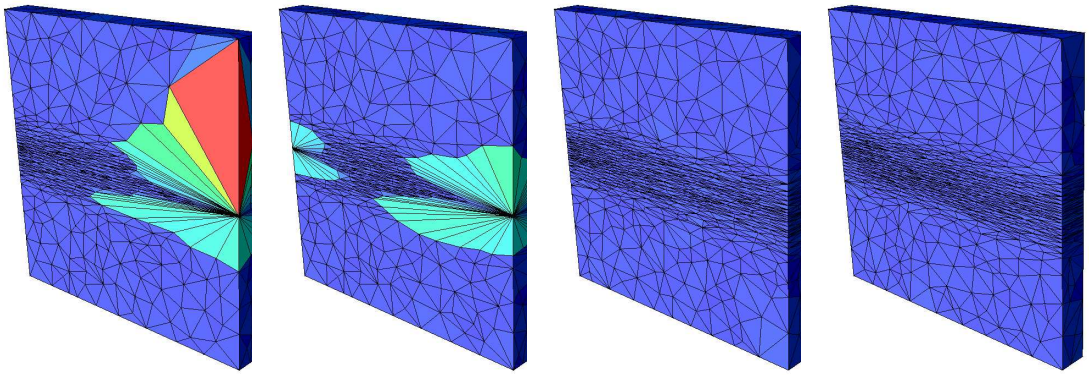
2.5.5 Sensitivity to the Initial Conditions

Ideally, a robust mesh optimization procedure should produce the same result independently of the initial grid, that is, of the initial conditions of the optimization process, since the desired grid properties are solely dictated by the target metric. This is, however, not true for the greedy scheme, particularly when the initial grid density is very far from the goal.

Various sizes of initial meshes are shown in Figures 31 and 32 for the box and the wing problems, respectively. The numbers at the bottom of each mesh refer to the total number of tetrahedra in the grid. Each initial mesh represents a different starting point in the solution domain, which typically might have many local minima. If the process starts from a point that is very close to the global minimum, it can easily end with a desired solution. On the other hand, if the process starts far from the target, entrapment in one of the local minima might be inevitable, as we have already witnessed several times. Here we investigate the ability of the LSA technique to escape from these local minima by comparing the effects of different initial triangulations on the final resulting meshes.

Tables 10 and 11 report the results for the isotropic and smooth ridge examples. It

appears that all initial conditions lead to similar results in terms of final mesh quality, (i.e., worst and average values of the merit function). Of course, since the routes taken towards the solution can be quite different in the various cases, the computational effort can vary substantially from one case to another. However, we can conclude that starting from an arbitrary point in the solution space, LSA will yield very similar end points given enough computation time. In contrast, greedy approach will get stuck in an unacceptable local minimum for initial points far away from the optimal point. Furthermore, the computational overhead of LSA is not prohibitive as shown in section 2.5.3. The situation is similar for the wing problem presented in Table 12, in which the algorithm again eventually reaches the target regardless of the starting point in the solution domain.



(a) $\theta_{init} = 0.0001$ (b) $\theta_{init} = 0.01$ (c) $\theta_{init} = 0.1$ (d) $\theta_{init} = 1000.0$

Figure 30: Smooth ridge problem with different initial temperatures

Table 5: Effect of different initial annealing temperatures for the isotropic cube problem using the LSA method.

θ_{init}	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
10^{-4}	0.28	0.080	18,973	1.00
10^{-3}	0.28	0.080	18,973	1.00
10^{-2}	0.28	0.080	18,965	0.99
10^{-1}	0.28	0.080	19,147	0.99
10^{0}	0.25	0.080	19,063	1.01
10^{1}	0.26	0.079	19,135	0.97
10^{2}	0.26	0.079	19,121	0.97
10^{3}	0.26	0.080	19,120	0.98
10^{4}	0.26	0.079	19,140	1.02

Table 6: Effect of different initial annealing temperatures for the smooth ridge problem using the LSA method.

θ_{init}	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
10^{-4}	12.07	0.260	6,465	1.00
10^{-3}	4.00	0.210	6,610	1.19
10^{-2}	4.56	0.230	6,531	1.76
10^{-1}	2.68	0.130	7,754	1.11
10^{0}	0.29	0.082	8,576	1.30
10^{1}	0.33	0.083	8,569	1.35
10^{2}	0.24	0.080	8,597	1.51
10^{3}	0.27	0.080	8,673	1.27
10^{4}	0.38	0.080	8,657	1.82

Table 7: Effect of different initial annealing temperatures for the wing problem using the LSA method.

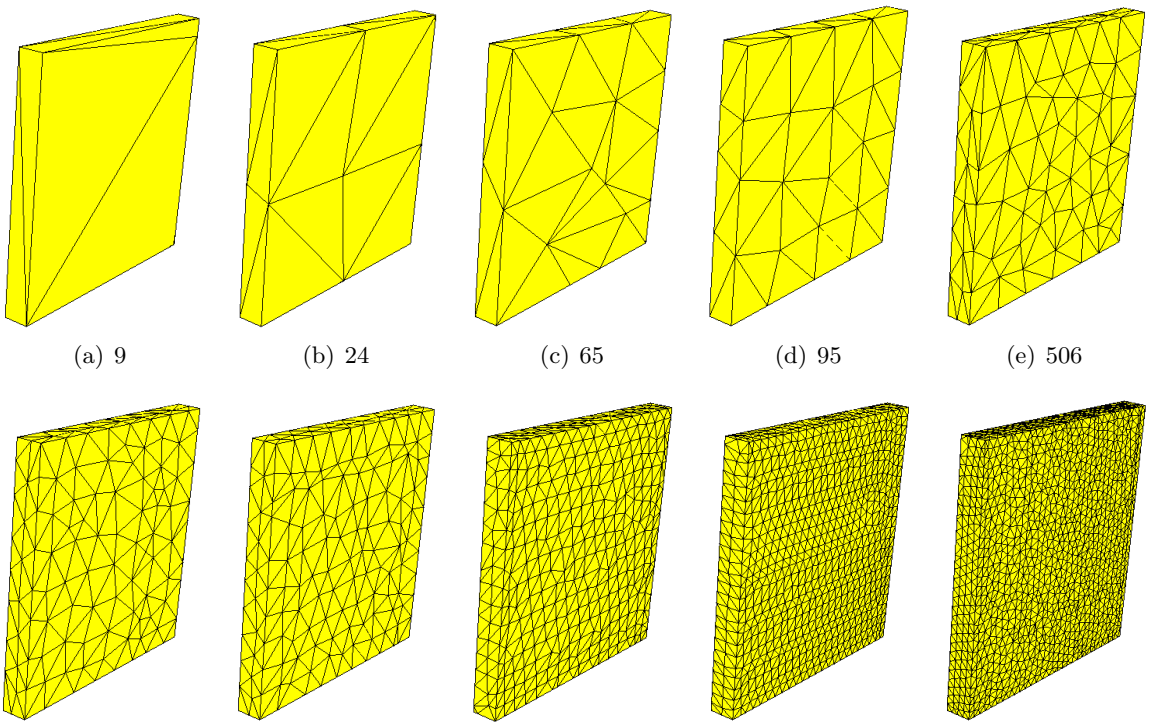
θ_{init}	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
10^{-4}	1.23	0.230	82,540	1.00
10^{-3}	0.35	0.081	101,917	4.02
10^{-2}	0.32	0.081	101,920	4.10
10^{-1}	0.29	0.081	101,945	3.71
10^{0}	0.27	0.081	101,880	4.27
10^{1}	0.29	0.081	101,897	3.64
10^{2}	0.28	0.081	101,645	4.49
10^{3}	0.27	0.081	101,821	4.26
10^{4}	0.27	0.081	101,575	4.30

Table 8: Effect of different values of $n_{\text{accept,max}}$ for the isotropic cube problem using the LSA method.

$n_{\text{accept,max}}$	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
2,500	0.24	0.079	18,666	97.21
1,500	0.23	0.078	18,907	63.96
750	0.21	0.080	19,096	41.17
150	0.21	0.080	18,619	9.30
50	0.27	0.079	18,747	3.04
20	0.27	0.078	18,709	1.49
10	0.23	0.079	18,568	1.09
5	0.25	0.078	18,692	1.03
3	0.23	0.080	18,775	1.07
1	0.27	0.080	18,804	1.00

Table 9: Effect of different values of $n_{\text{accept,max}}$ for the wing problem using the LSA method.

$n_{\text{accept,max}}$	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
50,000	0.27	0.081	101,819	4.50
5,000	0.29	0.081	101,741	4.40
1,500	0.26	0.081	101,292	4.76
750	0.29	0.081	101,918	0.98
300	0.24	0.081	101,838	0.99
150	0.22	0.081	101,831	1.00
50	0.22	0.081	101,672	1.00
20	0.22	0.081	101,641	1.15
10	0.22	0.081	101,847	1.00
5	0.22	0.081	101,823	1.00
3	0.28	0.081	101,921	1.04
1	0.29	0.081	101,892	1.00



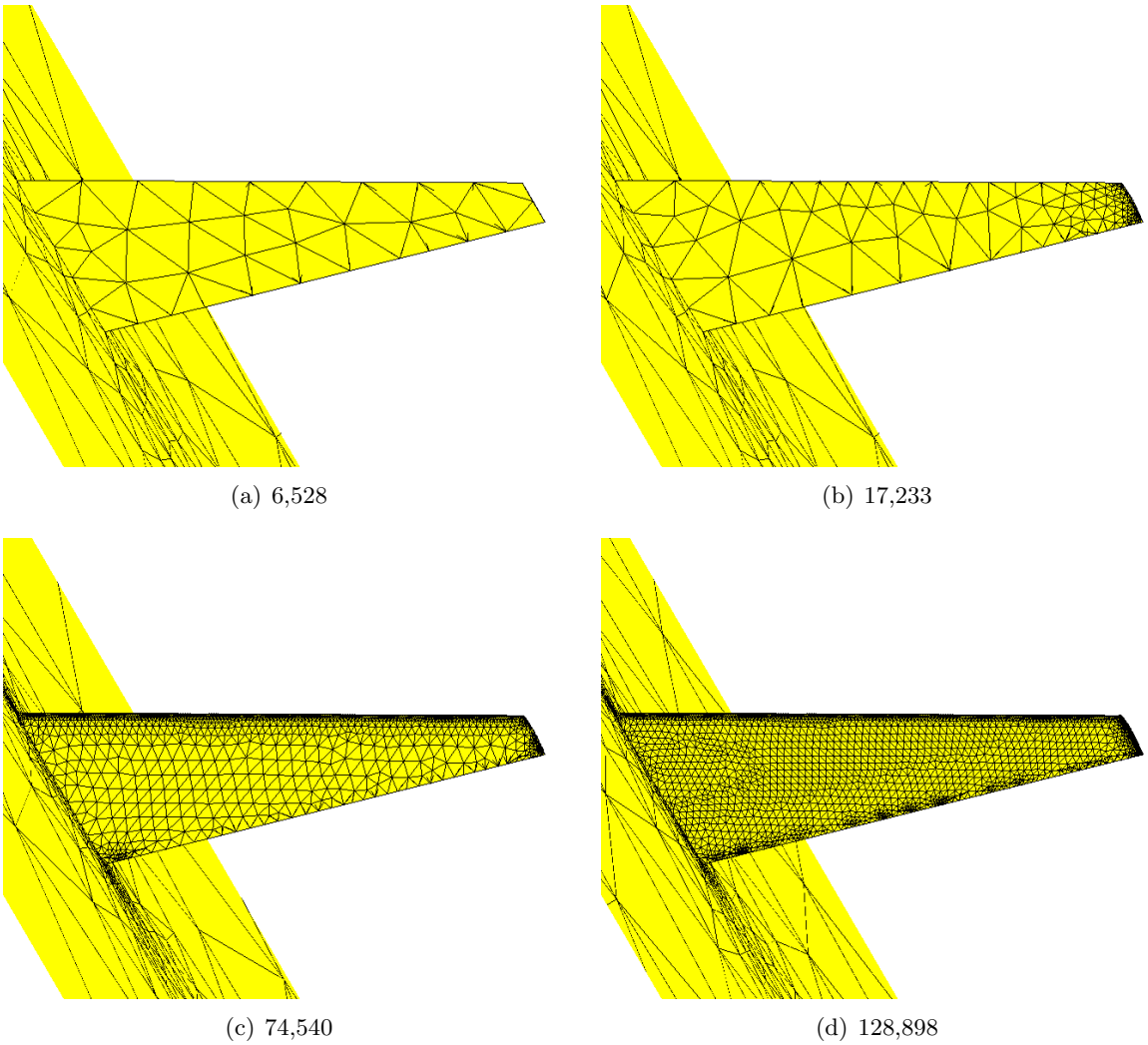
(a) 9 (b) 24 (c) 65 (d) 95 (e) 506

(f) 1031 (g) 1492 (h) 3101 (i) 7034 (j) 21595

Figure 31: Initial grids of different densities for the box problem.

Table 10: Optimized isotropic cube meshes using LSA for various initial grids.

Initial mesh size	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
9	0.28	0.080	13,227	1.00
24	0.29	0.081	13,093	0.85
95	0.26	0.082	13,096	0.63
506	0.26	0.081	13,047	0.83
1,031	0.28	0.082	12,999	0.57
1,492	0.27	0.082	13,001	0.54
3,101	0.29	0.084	12,872	0.19
7,034	0.15	0.077	11,745	0.21
21,595	0.16	0.031	21,146	0.05



(a) 6,528 (b) 17,233

(c) 74,540 (d) 128,898

Figure 32: Initial grids of different densities for the wing problem.

Table 11: Optimized anisotropic smooth ridge meshes using LSA with various initial grids.

Initial mesh size	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
9	0.29	0.083	8,502	1.00
24	0.35	0.083	8,662	0.45
65	0.29	0.084	8,623	0.32
95	0.27	0.086	8,542	0.34
506	0.27	0.087	8,591	0.39
1,031	0.33	0.090	8,454	0.17
1,492	0.30	0.088	8,393	0.26
3,101	0.24	0.078	8,636	0.53
7,034	0.23	0.077	10,998	0.78
21,595	0.28	0.086	10,110	0.27

Table 12: Optimized anisotropic wing meshes using LSA with various initial grids.

Initial mesh size	$J(\mathcal{T}_h)$	$A(\mathcal{T}_h)$	N	CPU/CPU$_{\text{ref}}$
6,528	0.27	0.080	97,469	1.00
17,233	0.33	0.082	100,610	0.72
74,570	0.27	0.082	101,480	0.61
86,541	0.33	0.077	97,765	0.64
128,898	0.33	0.080	104,676	0.81
210,102	0.33	0.065	130,437	0.82

Chapter III

DEFORMING MESHES WITH THE BALL-VERTEX METHODOLOGY

3.1 Introduction

For many numerical flow simulations, in order to accommodate the entire mesh to the imposed displacements on moving boundaries and interfaces, the use of smoothing or repositioning techniques is essential. For example, for airfoil oscillations and flutter predictions, either a body-conforming mesh has to be regenerated each time the boundaries move, or the existing grid needs to be deformed to comply with the imposed boundary displacements. The former option is quite cumbersome and computationally expensive, especially for three-dimensional problems. The latter option is preferred because it is easier to implement and makes mesh deformation more practical for unsteady problems. Nodal repositioning, or in general terms, mesh deformation, differs from previously studied h-type mesh optimization in the sense that it does not require topological modifications. The nodal connectivity and the quantities of elements remain unchanged. So deformation algorithms that take these factors into account have to be developed.

Studies in this area are few and limited in terms of their range of applications. The most commonly used technique is the classical spring analogy method, which works well as long as the given mesh is not very fine or the displacement requirement is not very large compared to the grid size. However, these conditions bring up a major problem, especially for implicit flow simulations, because the time steps allowed by the implicit solver are much larger so that the fluid mesh can undergo relatively large deformations.

Therefore, in this work, we study a new mesh deformation methodology capable of large displacements while avoiding computationally complex formulations. We are particularly interested in a unified algorithm that can deform any type of grid; hex, tet, prism, pyramid or pent including structural grids as well as hybrid meshes. The proposed method that

we refer to as "ball-vertex" is explained in detail in the following sections. We begin with the past studies in the area and talk briefly about spring analogy in section 3.4. Then we introduce collapse mechanisms and a means of controlling them with the ball-vertex method in section 3.7. Finally, we conclude with a variety of numerical examples ranging from a simple two-dimensional benchmark to a three-dimensional bending wing problem.

3.2 Literature Review

Deforming meshes are used in many applications, including problems involving moving boundaries or interfaces. In all these cases, the motion of a portion of the domain boundary is known, and one wants to deform the rest of the mesh in order to accommodate these imposed displacements.

The most widely used and simplest mesh deformation technique is spring analogy [14], in which each edge is replaced by a fictitious spring whose stiffness is inversely proportional to its edge length. Thus, longer edges will be softer while shorter ones will be stiffer, which may prevent the collision of the neighboring vertices. While the classical method performs reasonably well in a number of cases, it does indeed fail as soon as the local grid motion is not small compared to the local mesh size. Unfortunately, in many practical cases, the necessary grid displacements are not small. Furthermore, even if the displacements are small, the spring analogy or the edge-spring method can not prevent the creation of flat elements. In the presence of nearly flat elements, edge-vertex (in two dimensions) or face-vertex (in three dimensions) cross-overs are commonly observed. Clearly, any cross-over will lead to a locally invalid grid, and therefore must be avoided.

In fact there is a need for methods that can specifically deal with large deformation problems. To address this issue, torsional springs were added to the linear edge springs by Farhat et al. [34, 31] in order to avoid the possible collapse mechanisms of triangles and tetrahedra. The possible face collapse mechanism is prevented by controlling the face area in two dimensions. In three spatial dimensions, additional virtual triangular faces are added to each tetrahedron. Each of these additional faces is equipped with torsional springs at its vertices that oppose the possible face collapse mechanism. In turn, the inserted faces oppose

the motion of each vertex of the tetrahedron towards its opposite face, thereby controlling the collapse mechanism of the tetrahedron. The combination of linear and torsional springs can significantly improve the robustness of the spring analogy method, but it cannot be used in general cased, due to the possibility of distorted quadrilateral and hexahedral elements [12]. Another obstacle to using this method is that it can be applied only to simplicial or tetrahedral meshes unless the hexahedral/prism/pyramid type elements are divided into some tet combinations. In addition, it's been shown in reference [22] that torsional spring method still fails because it either creates flat elements or fails to achieve full displacement required.

The torsional springs method as well as the classical spring analogy are discrete deformation techniques. More often than not, a PDE describing the mesh motion can be coupled with the governing equations of the problem [55, 25, 24, 11, 73, 17, 7, 29, 92, 10]. However, this continuous type of deformation techniques are difficult to handle for problems with more complex geometries because not only does the number of unknowns increase but the construction of the governing equations also becomes relatively harder. An approach that decouples the moving mesh equations from the governing equations can make the methodology easier to implement in a broad range of applications.

In this work, we apply a new, simpler method of controlling collapse mechanisms in unstructured quadrilateral/hexahedral and triangular/tetrahedral meshes. This method is based on the idea of complementing the linear edge springs with linear face-vertex springs in three dimensions or linear edge-vertex springs in two dimensions. These additional springs effectively constrain each vertex within the polyhedral ball that encloses it, contrasting the possible collapse mechanisms of the grid elements. Furthermore, the presence of additional springs is also beneficial in terms of mesh quality since these springs tend to keep each vertex close to the centroid of the ball, pushing it away from its boundaries.

3.3 Deforming Meshes

Let us consider the mesh deformation problem, in which a portion of the mesh boundary moves, and the grid is deformed to accommodate the displacements of the boundaries. For

this purpose a vertex repositioning problem must be solved which is robust and efficient, so that it does not generate invalid elements, even for large amplitudes of motions. The basic idea behind all methods for this class of problems is to define a suitable fictitious elasticity problem over the domain.

The fictitious problem can either be continuous [83] or discrete. For the former, partial differential equations are discretized in space, for example, by using finite element methods. Alternatively, a lumped-parameter discrete structural model can be used. In this study, we consider the discrete case, in which the fictitious problem defines a suitable network of springs associated with the unstructured mesh of quads or hexahedra. The problem is then transformed into the construction of the best possible network of springs that a) is inexpensive to compute; b) does not contain collapse mechanisms; and c) leads to graded and well-shaped deformed grids, even for large imposed displacements.

We consider a deforming mesh problem, i.e., a problem in which we are interested in deforming the domain Ω, and hence the grid $\mathcal{T}_h$ associated with it, in order to accommodate some given displacement on a portion of its boundary. The deformed configuration of the elastic domain can then be computed under the action of the driving displacements. In general, the domain boundary can be partitioned according to the following criterion [22]:

$$\Gamma = \Gamma_m + \Gamma_0 + \Gamma_s \tag{46}$$

The moving boundary of the domain is denoted as Γ_m, and displacements on Γ_m are known such that

$$\boldsymbol{u}_i = \boldsymbol{g} \quad \forall i \in \{V\}_{\Gamma_m} \tag{47}$$

where $\boldsymbol{u}_g$ is the prescribed vertex displacements. The portion of the boundary where no displacement is imposed can be subdivided into two parts, Γ_0 and Γ_s, which correspond to stationary and sliding boundaries, respectively:

$$\boldsymbol{u} = 0 \quad on\ \Gamma_0 \qquad \boldsymbol{u} \cdot \boldsymbol{n} = 0 \quad on\ \Gamma_s \tag{48}$$

where $\boldsymbol{n}$ indicates the local normal. Far-field boundary is an example of Γ_0 and symmetry planes represent a form of Γ_s. In the simulation of a transient process, the driving boundary conditions for the mesh deformation problem are to be regarded as functions of time.

Consequently, a mesh deformation problem is solved at each time step during the transient simulation. Imposing the boundary conditions on Γ_0 and Γ_s, yields the following fictitious elasticity problem

$$\boldsymbol{M}\ddot{\boldsymbol{u}}_M + \boldsymbol{C}\dot{\boldsymbol{u}}_M + \boldsymbol{K}\boldsymbol{u}_M = \boldsymbol{B}\boldsymbol{g}, \tag{49}$$

where $\boldsymbol{u}_M$ is the vector of displacements of the moving vertices of the grid, $\boldsymbol{g}$ is the vector of imposed displacements on Γ_m, $\boldsymbol{M}$, $\boldsymbol{C}$, $\boldsymbol{K}$ are the inertia, damping, and stiffness matrices, respectively, and $\boldsymbol{B}$ is found by imposing the boundary conditions on the various portions of the domain boundary. More often than not, and in the present work, only the static version of the problem is used, i.e., one sets the time derivatives of $\boldsymbol{u}_M$ to zero and solves

$$\boldsymbol{K}\boldsymbol{u}_M = \boldsymbol{B}\boldsymbol{g}. \tag{50}$$

This linear system can be solved with either direct or indirect methods.

3.4 Basic Spring Analogy

We briefly review the classical edge spring method. Given two vertices i and j, the edge-vector from i to j is defined as

$$\boldsymbol{e}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i \tag{51}$$

and its length is $L_{ij} = \sqrt{\boldsymbol{e}_{ij}\boldsymbol{e}_{ij}}$ with the unit edge vector given as $\boldsymbol{i}_{ij} = \frac{\boldsymbol{e}_{ij}}{L_{ij}}$. The displacements of vertices i and j are noted as $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$, respectively. The resulting force on vertex i along the unit vector can be evaluated as

$$\boldsymbol{f}_{ij}^{Edge} = k_{ij}(\boldsymbol{u}_j - \boldsymbol{u}_i)\boldsymbol{i}_{ij}\boldsymbol{i}_{ij} = \boldsymbol{f}_{ji}^{Edge} \tag{52}$$

where k_{ij} is the spring stiffness, typically chosen as inversely proportional to the edge length. Thus, the small elements tend to be very stiff while larger elements are softer [14].

$$k_{ij} = \frac{1}{L_{ij}} \tag{53}$$

The position of each vertex is found by writing its equilibrium under the effect of all its n_E edge-connected springs [14]

$$\sum_{j=1}^{n_E} \boldsymbol{f}_{ij}^{Edge} = 0 \tag{54}$$

The coefficient matrix (i.e., stiffness matrix) for the resulting linear equation system is an assembly of 3x3 block entry contributions due to the edge connecting vertices i and j. These block entries, $\boldsymbol{K}_{ii} = -\boldsymbol{A}_{ij}$, $\boldsymbol{K}_{ij} = \boldsymbol{A}_{ij}$, $\boldsymbol{K}_{ji} = \boldsymbol{K}_{ji}$, $\boldsymbol{K}_{jj} = -\boldsymbol{A}_{ij}$, where $\boldsymbol{A}_{ij} = k_{ij}\boldsymbol{i}_{ij}\boldsymbol{i}_{ij}$, are readily computed by the inspection of equation (52). The distribution of the local stiffness matrices in the global stiffness matrix is provided in the following equation

$$\begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \boldsymbol{K}_{ii} & \dots & \dots & \boldsymbol{K}_{ij} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \boldsymbol{K}_{ji} & \dots & \dots & \boldsymbol{K}_{jj} & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \boldsymbol{u}_i \\ \vdots \\ \boldsymbol{u}_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \vdots \\ \mathbf{0} \\ \vdots \\ \vdots \end{bmatrix} \tag{55}$$

The most common method in literature for solving this system is an explicit Gauss-Seidel; however, in this work we implement an implicit preconditioned conjugate gradient method (PCG) [13, 77].

3.5 Controlling Collapse Mechanism

Prevention of entanglement is crucial in mesh deformation algorithms. The classical spring analogy method [14] contains collapse mechanisms that can lead to entanglements. For example when a vertex crosses a neighboring mesh face, as shown in Figure 33, degeneracy is inevitable. A simple illustration of the generation of a degenerate element is shown in the figure, in which a large displacement is applied on the left-top vertex i, and then the vertex moves under the effect of this imposition. The elements with red solid boundaries are topologically invalid and can not be accepted. This is usually the case when a large displacement is applied with the edge spring method; therefore, the algorithm can be used for only moderate displacements. An improved method based on the use of additional torsional springs at the vertices was proposed by Farhat et al. [34, 31] for two-dimensional grids. The torsional springs were designed to guarantee that neighboring springs could not interpenetrate each other, and hence preventing mesh entanglement. However, the introduction of torsional springs renders the problem non-linear, so a linearized version of

the scheme must be developed. Furthermore, the method becomes quite cumbersome in three dimensions.

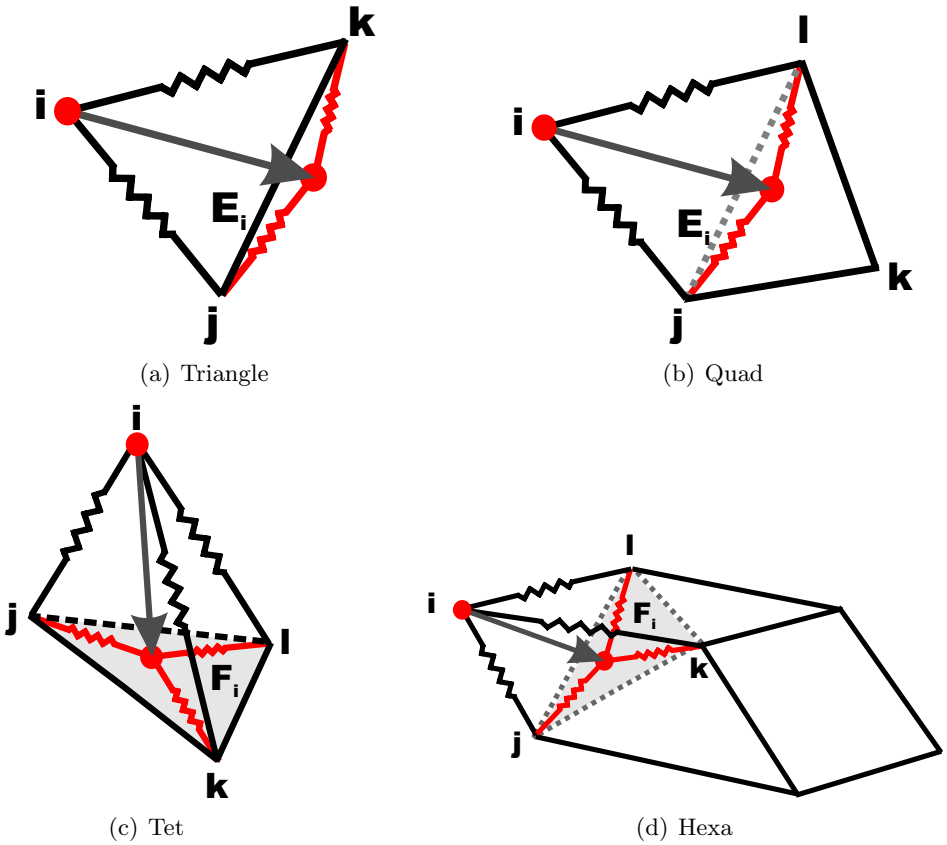


Figure 33: Collapse mechanisms with finite edge-spring stretching in two dimensions (top row) for a triangle (a), a quad (b), and in three dimensions (bottom row) for a tetrahedron (c), and a hexahedron (d).

We use the new, simple technique "ball-vertex" for quads/hexahedra unstructured as well as triangular/tetrahedral meshes. Ball-vertex guarantees valid meshes, even for large amplitudes of displacements. The basic idea is to use the classical edge spring method of Batina [14] together with additional linear springs, which provides increased stiffness to the system, such that, they oppose the creation of invalid elements.

The torsional springs method is formulated only for tet mesh applications. However, in many numerical simulations, hex meshes are preferred over tets because of their three basic features [80]. First, studies have shown that the hex meshes can be more accurate for a given number of degrees of freedom [28]. Second, for a given characteristic edge length and similar analysis accuracy, hex meshes are computationally more efficient. Various examples

record a four- to ten-time improvement in computational performance of hex meshes in such situations. This is a significant difference especially when computing resources are limited. Lastly, a hex mesh, by mimicking the structure of the geometry being modelled, can be useful for special applications such as viscous flows in CFD.

3.6 Collapse Mechanisms of Simplicial and Non-Simplicial Elements

The classical edge-spring method contains collapse mechanisms for both simplicial and non-simplicial elements.

Consider first the two-dimensional case, and specifically the triangle of Figure 33(a). Vertex i is initially in the half left plane of its facing mesh edge E_i, and the triangle is valid according to for example, a counter-clockwise numbering rule of its bounding vertices. The vertex can now be displaced to the right half plane with only finite stretching of the edges connecting it to the boundary vertices of E_i, as shown in the figure. The element is now invalid according to the numbering rule and has a negative area.

Similarly, consider the quad depicted in Figure 33(b). Even in this case, one can displace vertex i to the point of crossing its facing diagonal E_i with only finite stretchings of its connected edges. Notice that E_i is not a mesh edge in this case, but a diagonal edge that has as bounding vertices, the two mesh vertices that are edge-connected to vertex i. The angle between the two edges of the quad connected to i is now larger than π, and the element is again invalid (concave).

Consider now the three-dimensional case. A tetrahedron of a simplicial mesh can be transformed into a flat element, again with only a finite stretching of its edges; this situation is depicted in Figure 33(c). In fact, consider vertex i and its opposite mesh face F_i. The vertex is initially in the half space pointed by the oriented normal to F_i, and hence, the tetrahedron is valid. The vertex is now displaced to the other half space, crossing the plane of F_i. The element is now of negative volume, and hence invalid, but the crossing of the F_i plane has been accomplished with finite stretchings of the edges connecting i with the bounding vertices of F_i.

Finally, consider the hexa depicted in Figure 33(d), and more specifically, its vertex i

and its opposite face F_i. Notice that in this case F_i is not a mesh face, but a triangular face that has as bounding vertices, the three vertices that are edge-connected to i. Again, vertex i can be displaced to the other half space defined by the plane of F_i with only finite stretchings of its edges. When this happens, the dihedral angles formed by the element faces at the three edges connected to i become larger than π, and the element becomes invalid (concave).

It is interesting to observe that the condition for a valid mesh to remain valid after deformation is the same for simplicial and non-simplicial meshes in two and three spatial dimensions. This condition is expressed in terms of the *ball* of a mesh vertex. The ball of a vertex is defined as the polyhedral cavity bounded by all edges (in two dimensions) or triangular faces (in three dimensions) that are edge-connected to the vertex. The ball of vertex i in two spatial dimensions is shown in Figure 35(a) for the case of a simplicial mesh; its bounding edges are labeled E_1 through E_5. The ball of vertex i for a two-dimensional, non-simplicial grid is shown in Figure 35(b) and (d). Notice that in the non-simplicial case, the bounding entities of the ball are not part of the original mesh; thus, the bounding edges E_1, E_2, and E_3 of the ball of i are shown using dotted lines in the figure. The ball of vertex i for a tetrahedral grid is shown in Figure 35(c), while the ball of i for a hexahedral grid is depicted in Figure 35(d). Similarly, in this case, the bounding edges are not entities of the mesh, and are represented using dotted lines in the figure.

Both in two and three dimensions, both for simplicial and non-simplicial meshes, an element becomes invalid if one of its vertices leaves its associated ball, which for simplicial meshes, will cause either the area (in two dimensions) or the volume (in three dimensions) of the element to change sign and for non-simplicial meshes, it will cause the element to become concave, i.e., to have one angle (or dihedral angles in three dimensions) that exceeds π.

3.7 Controlling Collapse Mechanisms with Ball-Vertex Springs

The construction of the ball-vertex springs for simplicial meshes was first discussed in Reference [22]. The simplicial and non-simplicial two- and three-dimensional cases are illustrated

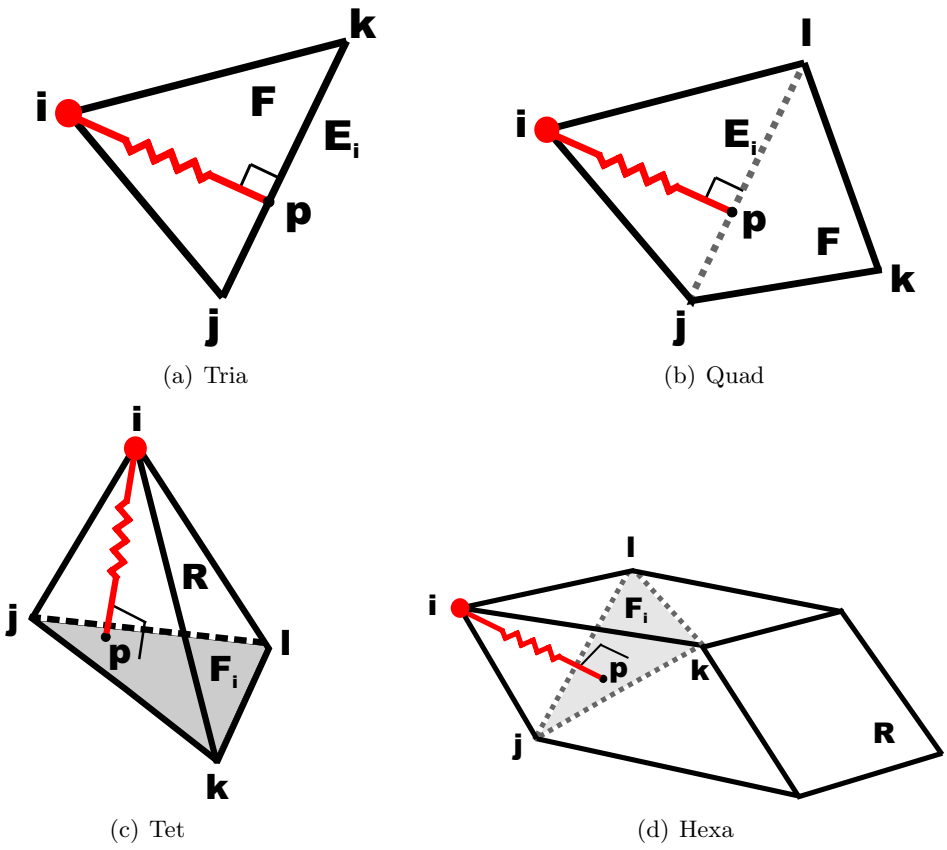
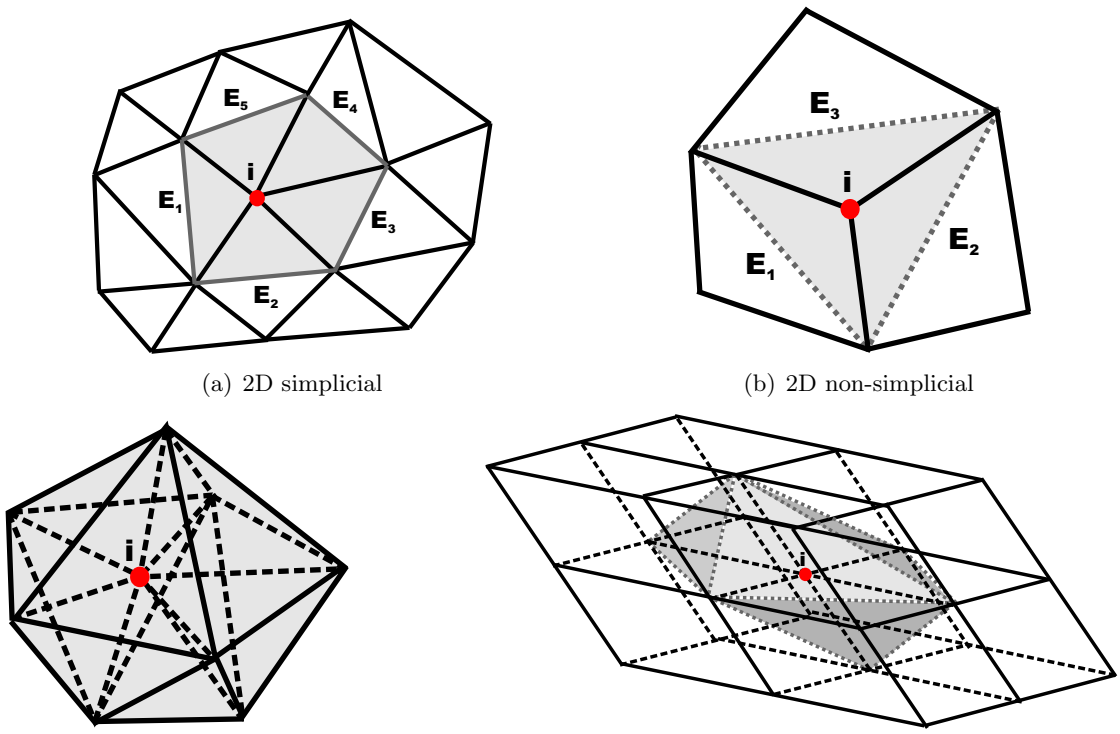


Figure 34: Ball-vertex springs for 2D (top row) tria (a), quad (b), and 3D (bottom row) tetrahedron (c) and hexahedron (d).

in Figure 34.

In two spatial dimensions, we introduce linear springs that resist the motion of a mesh vertex towards its opposite edge. The case of a triangle is illustrated in Figure 34(a), and the edge E_i opposite i is a mesh edge. The case of a quad is illustrated in Figure 34(b), and in this case the edge opposite i is the diagonal edge shown in the figure using a dotted line. In summary, in the two dimensional case, for each mesh face F using a vertex i, a linear spring that connects i with its projection point p on the edge E_i of the ball of i is constructed.

The situation is analogous in three dimensions. Specifically, the case of a tetrahedron is shown in Figure 34(c), while the case of a hexahedron is depicted in Figure 34(d). More precisely, for each region R using a vertex i, a linear spring that connects i with its projection point p on the triangular face F_i of the ball of i is constructed. Notice that in both the two-



(a) 2D simplicial

(b) 2D non-simplicial

(c) 3D simplicial

(d) 3D non-simplicial

Figure 35: Balls of vertex i in two dimensions (top row) for the simplicial (a) and non-simplicial (b) cases, and in three dimensions (bottom row) for the simplicial (c) and non-simplicial (d) cases.

and three-dimensional cases, the projection points p might fall outside of the edge or face, respectively.

Once the additional springs are constructed for all elements using vertex i, one has effectively constrained the same vertex from leaving the polyhedral ball that encloses it. The position of each mesh vertex is found by writing its equilibrium under the effect of its ball-vertex springs. To this effect, consider the face-vertex springs depicted in Figure 34. If $\boldsymbol{u}_i$ and $\boldsymbol{u}_p$ denote the displacements of vertex i and of its projection point p on E_i or F_i, the resulting forces on i and p can be expressed as

$$\boldsymbol{f}_{ip} = k_{ip}(\boldsymbol{u}_p - \boldsymbol{u}_i) \cdot \boldsymbol{i}_{ip}\boldsymbol{i}_{ip} = -\boldsymbol{f}_{pi}, \tag{56}$$

where $\boldsymbol{i}_{ip}$ is the unit vector along the spring calculated as $\boldsymbol{i}_{ip} = \boldsymbol{e}_{ip}/L_{ip}$ with

$$\boldsymbol{e}_{ip} = \boldsymbol{x}_p - \boldsymbol{x}_i, \tag{57}$$

and where the edge length is denoted by $L_{ip} = \sqrt{\boldsymbol{e}_{ip} \cdot \boldsymbol{e}_{ip}}$. Furthermore, k_{ip} is the spring

stiffness; in this work, we use a spring stiffness that is inversely proportional to the edge length, $k_{ip} = 1/L_{ip}$, although clearly other choices are possible.

The only substantial difference with respect to the edge spring case $\boldsymbol{u}_p$ is given by the fact that $\boldsymbol{u}_p$ is now the displacement of a virtual point and not of an existing vertex. The displacement of the projection point p is obtained by interpolating the displacements of the vertices of either its edge E_i or its face F_i. In the latter, the position $\boldsymbol{x}_p$ of point p is computed as the normal projection of i on F_i as

$$\boldsymbol{x}_p = \boldsymbol{x}_i - (\boldsymbol{x}_i - \boldsymbol{x}_j) \cdot \boldsymbol{n} \cdot \boldsymbol{n}, \tag{58}$$

where

$$\boldsymbol{n} = \frac{\boldsymbol{i}_{jk} \times \boldsymbol{i}_{jl}}{\|\boldsymbol{i}_{jk} \times \boldsymbol{i}_{jl}\|} \tag{59}$$

is the unit normal to face F_i. Given $\boldsymbol{x}_p$, the interpolation coefficients ξ and η corresponding to p can be computed such that

$$\boldsymbol{x}_p = \xi \boldsymbol{x}_j + \eta \boldsymbol{x}_k + (1 - \xi - \eta)\boldsymbol{x}_l. \tag{60}$$

No special action is required if the projected point falls outside the targeted face. Solving for ξ and η renders

$$\xi = \frac{(\boldsymbol{x}_j - \boldsymbol{x}_l) \cdot (\boldsymbol{x}_p - \boldsymbol{x}_l)}{\|\boldsymbol{x}_j - \boldsymbol{x}_l\|}, \tag{61a}$$

$$\eta = \frac{(\boldsymbol{x}_k - \boldsymbol{x}_l) \cdot (\boldsymbol{x}_p - \boldsymbol{x}_l)}{\|\boldsymbol{x}_k - \boldsymbol{x}_l\|}. \tag{61b}$$

Finally, given ξ and η as computed above, the interpolated displacement at p is obtained as

$$\boldsymbol{u}_p = \xi \boldsymbol{u}_j + \eta \boldsymbol{u}_k + (1 - \xi - \eta)\boldsymbol{u}_l. \tag{62}$$

In a similar manner, the spring force applied to point p is linearly distributed among the vertices $j, k,$ *and* l of ball face F_i (Figure 34(c) or (d)). The virtual work of the spring between i and p due to the infinitesimal variation in the position of its two end points is

$$\delta W_{ip} = -\boldsymbol{f}_{ip} \cdot \delta \boldsymbol{u}_i - \boldsymbol{f}_{pi} \cdot \delta \boldsymbol{u}_p, \tag{63}$$

where $\delta \boldsymbol{u}_p$ is the virtual variation of equation (62). Consequently, the forces applied by the spring on vertices j, k and l of the ball face F_i are

$$\boldsymbol{f}_{ip,j} = \xi \boldsymbol{f}_{ip}, \tag{64a}$$

$$\boldsymbol{f}_{ip,k} = \eta \boldsymbol{f}_{ip}, \tag{64b}$$

$$\boldsymbol{f}_{ip,l} = (1 - \xi - \eta) \boldsymbol{f}_{ip}. \tag{64c}$$

The position of each vertex is found by writing its equilibrium under the effect of all its incident spring forces. To this end, one first writes the stiffness matrix $\boldsymbol{K}_{ip}$ due to the insertion of one spring; by using the principle of virtual work (63), $\boldsymbol{K}_{ip}$ is readily found as

$$\boldsymbol{K}_{ip} = k_{ip} \begin{bmatrix} \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & -\xi\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & -\eta\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & -(1-\xi-\eta)\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T \\ & \xi^2 \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & \xi\eta\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & \xi(1-\xi-\eta)\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T \\ & & \eta^2 \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T & \eta(1-\xi-\eta)\ \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T \\ & \text{symm.} & & (1-\xi-\eta)^2 \boldsymbol{i}_{ip}\boldsymbol{i}_{ip}^T \end{bmatrix}. \tag{65}$$

Then the entries of the global stiffness matrix due to the presence of the virtual edge ip is obtained by the following distribution of the individual contributions,

$$\begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \boldsymbol{K}_{ii} & \cdots & \boldsymbol{K}_{ij} & \cdots & \boldsymbol{K}_{il} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \boldsymbol{K}_{ji} & \cdots & \boldsymbol{K}_{jj} & \cdots & \boldsymbol{K}_{jl} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \boldsymbol{K}_{li} & \cdots & \boldsymbol{K}_{lj} & \cdots & \boldsymbol{K}_{ll} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ \boldsymbol{u}_i \\ \vdots \\ \boldsymbol{u}_j \\ \vdots \\ \boldsymbol{u}_l \\ \vdots \end{bmatrix} = \begin{bmatrix} \\ \vdots \\ \vdots \\ \boldsymbol{0} \\ \vdots \\ \vdots \\ \end{bmatrix} \tag{66}$$

where each $\boldsymbol{K}_{ij}$ represents a 2x2 block matrix in two dimensions or a 3x3 block matrix in three dimensions.

Finally, the contributions of edge springs (eqn. 55) and face springs (eqn. 66) are summed to find the resultant global coefficient matrix. The ball-vertex springs can clearly be used alone instead of in combination with the classical edge springs. Imposing the boundary conditions, one finds the solution of the system (50) for displacements $\boldsymbol{u}_i$. In

this work, we use a preconditioned conjugate gradient method [8]. Conjugate gradient algorithms are the most prominent methods for solving such type of sparse system of linear equations. Although they are more complex than classical relaxation methods, the use of such algorithms contributes to appreciable computational savings. However, it is generally agreed that a preconditioner is necessary for large-scale applications in order to accelerate the convergence of the method. For this problem, we have employed incomplete Cholesky preconditioning [39], which speeds up the solution by improving the condition number of the coefficient matrix.

3.8 Numerical Examples and Results

In the following sections, five deformation problems are presented for a benchmark, a pitching-plunging airfoil, a pitching-plunging wing, a bending wing, and a multi-element airfoil.

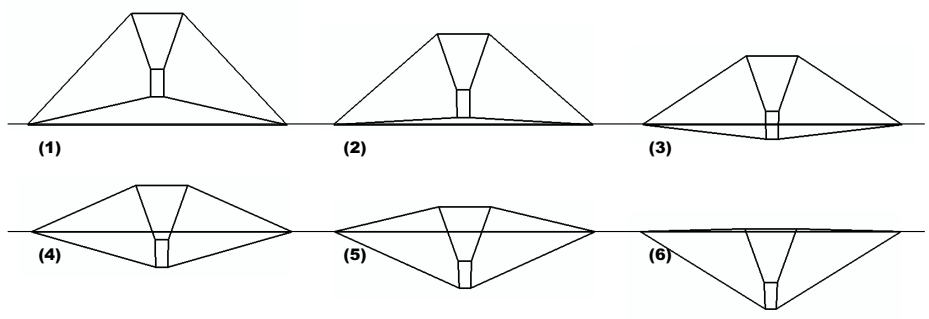
3.8.1 Benchmark Problem

The benchmark problem, which deals with an unstructured non-simplicial mesh subjected to large imposed displacements, was inspired by the benchmark problem defined by Farhat et al. [34] for the torsional springs method applied to triangular elements. Figure 36 illustrates a two-dimensional version of the problem, with the plain edge-spring method results shown in Figure 36(a) and the ball-vertex method results in Figure 36(b). The initial configuration is labeled (1), and the mesh is made of five elements. Large vertical displacements are applied at the top two vertices, while the bottom two remain fixed. The total imposed displacement is just slightly smaller than the height of the domain, and it is applied in five incremental steps.

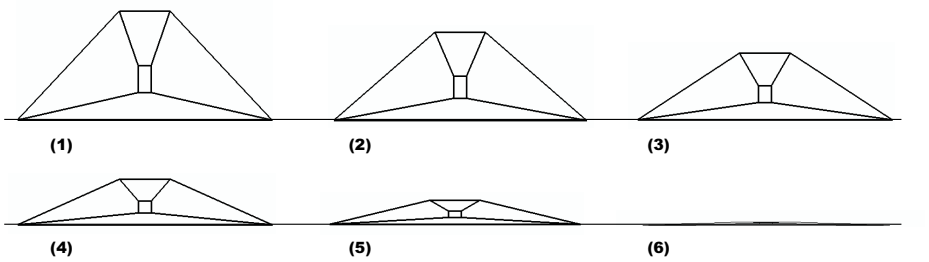
We assess and compare the two techniques for this particular problem by simply checking whether or not they produce invalid mesh configurations [34]. In Figure 36(a), we observe that with the edge-spring method, invalid quads are formed almost immediately when the central free vertices of the grid cross the fixed bottom edge at the second displacement step. However, the network generated with the ball-vertex method remains valid at all times, squeezing the elements but keeping each vertex within its ball during the entire motion.

Figure 37 illustrates the three-dimensional version of the problem. In this case, the mesh is made of six hexahedral elements. Large vertical displacements are applied to the four top vertices while the bottom four remain fixed.

The results obtained with the plain edge-spring method are presented in Figure 37(a), while the ones of the ball-vertex method in Figure 37(b). Even in this case, while the edge-spring method produces invalid elements from the very beginning, the ball-vertex method is capable of yielding a valid grid, even for imposed displacements only slightly smaller than the initial height.



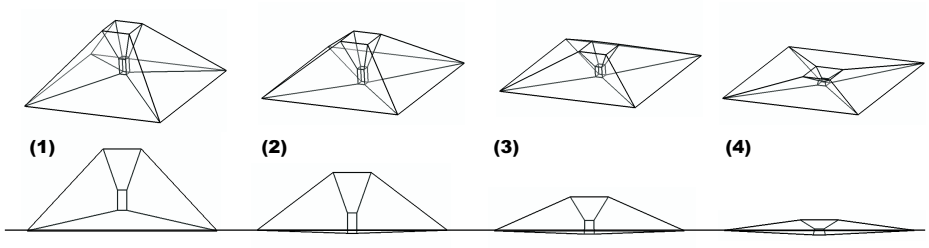
(a) Edge-spring method



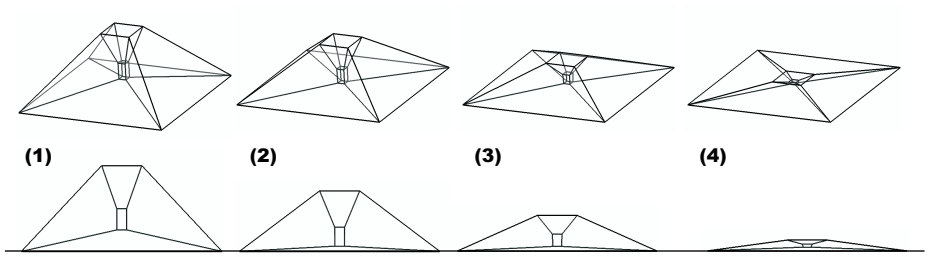
(b) Ball-vertex method

Figure 36: Two-dimensional benchmark problem with the edge-spring (a), and the ball-vertex method (b).

If the amplitude of the motion decreases significantly, we may get valid quadrilateral meshes even with an edge spring analogy as well. However, the amplitude has to be decreased to the order of $h/100$; in other words, the total number of incremental steps has to be increased to 100, so that entanglement does not result for this example problem.



(a) Edge-spring method



(b) Ball-vertex method

Figure 37: Three-dimensional benchmark problem with the edge-spring (a), and the ball-vertex method (b).

3.8.2 Oscillating Airfoil Problem

In this section, we consider a pitching and plunging airfoil of a unit chord with 996 quads and 937 vertices, which moves according to equation (67) with a pitching amplitude of $\theta_0 = 30^o$ and a plunging amplitude of $h_0 = 0.22\ chords$.

$$\begin{aligned} \theta_n &= \theta_0 sin(2\pi\, n/N) \\ h_n &= h_0 sin(2\pi\, n/N) \end{aligned} \tag{67}$$

n, in the equation, stands for the generic time step, while N stands for the total number of time steps per cycle. The simulation is conducted for a total of five cycles for which preconditioned conjugate gradient convergence criterion is chosen as $\epsilon = 10^{-5}$ and N as 100.

While Figure 38 exhibits upstroke and downstroke mesh configurations of the airfoil at the end of the first cycle, Figure 39 represents mesh quality plots of this motion. The first plot shows the history of the minimum face spring length i.e., $min_{L_{ip}}$ with respect to time steps, whereas the second plot in the figure shows a "compliance residual" computed in Frobenius norm [21]. Compliance residual is a measure that quantifies the deviation from

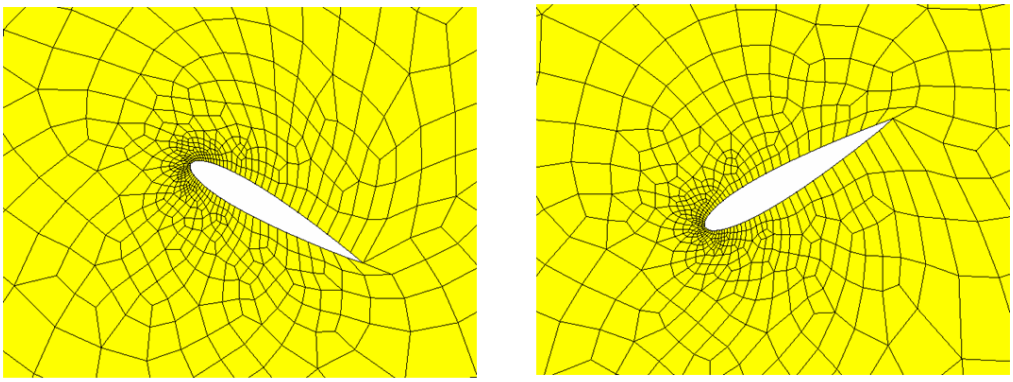


Figure 38: Pitching and plunging airfoil with mesh size of 996 quads and 937 vertices, using ball-vertex method.

the initial mesh, and computed by the following metric-based distortion function [21]:

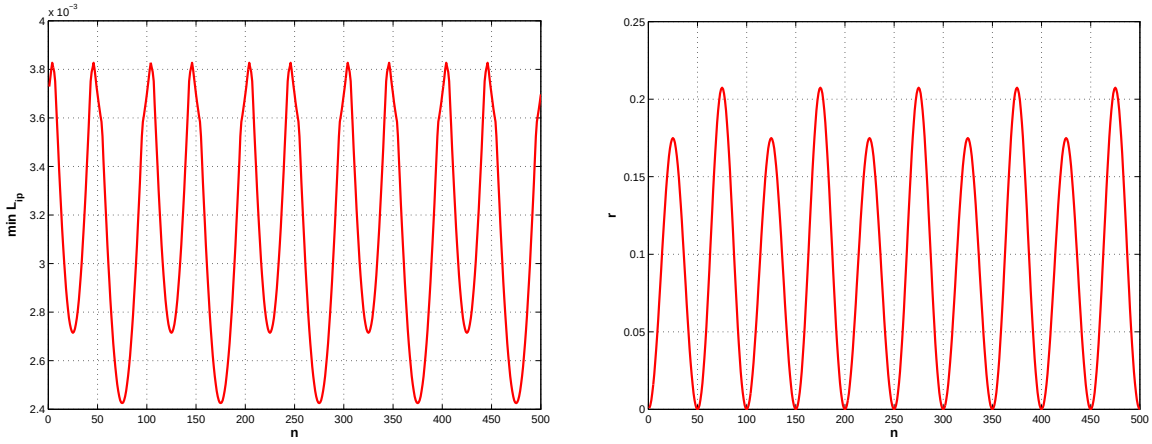
$$r = \frac{1}{n_K} \sum_K \|(\boldsymbol{M}_K - \boldsymbol{M}_K^{\text{initial}})(\boldsymbol{M}_K^{-1} - \boldsymbol{M}_K^{\text{initial}^{-1}})\|, \tag{68}$$

where $\boldsymbol{M}_K$ is the current metric of element K, $\boldsymbol{M}_K^{\text{initial}}$ is its initial value in the undeformed configuration, n_K is the number of elements in the grid, and $\|\cdot\|$ represents the Frobenius norm. The metric of a simplicial element is constant over the element itself, but this is not the case for a non-simplicial element. Therefore, the metrics appearing in equation (68) were evaluated by splitting each quad into two triangles by computing the metric of each triangle and averaging the individual contributions. Similarly, in three dimensions, a hexahedron needs to be split into a possible tetrahedra configuration before the averaging.

As the mesh is deformed, the metric of each element departs from its initial value because the element is stretched and rotated. It would be desirable to verify that, as the airfoil returns to its zero angle-of-attack position at the end of each cycle, each element in the grid returns to its original configuration, since this would imply that a full cycle of deformation has not changed the initial grid. This effect can be assessed by plotting function r, which fully captures the complete state of deformation of each element, versus time. There exists no standard, with regard to the value of the metric r. Its value may vary with the nature of the motion imposed, e.g., the pitching amplitude or the complexity of the problem. If r has a large value, this indicates that a large deformation is taking place, it is normal as long as a degenerate element is not generated ($r = \infty$). On the other

hand, $r = 0$ is an important value that indicates that the current mesh is identical with the initial mesh, which is highly desirable at the end of each full cycle because it shows that the deformation method is robust and does not damage the original mesh. This feature is crucial, especially for periodical problems (e.g., flutter).

Therefore, it is important to notice in Figure 39 that the ball-vertex technique recovers completely the original mesh configuration at rest whenever the airfoil passes through its original position, i.e., the zero angle-of-attack, which corresponds to time steps 0, 100, 200..,500 in this case. So, the mesh is not damaged as a result of the airfoil motion i.e., compliance residuals at zero-angle-of-attack positions are zero. This is also confirmed by a closer view of the leading edge after the fifth cycle in Figure 40(b).

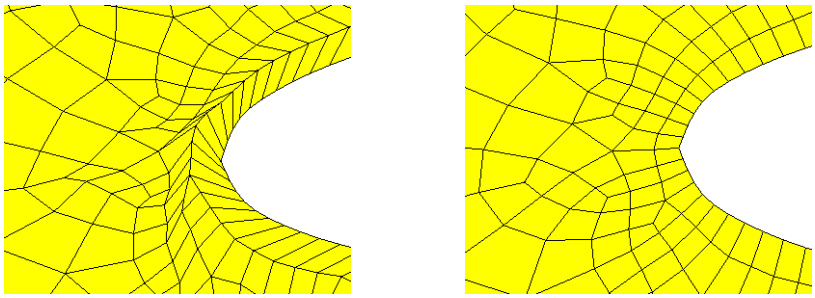


(a) (b)

Figure 39: Ball-Vertex: Quality plots for pitching and plunging airfoil with 996 quads and 937 vertices.

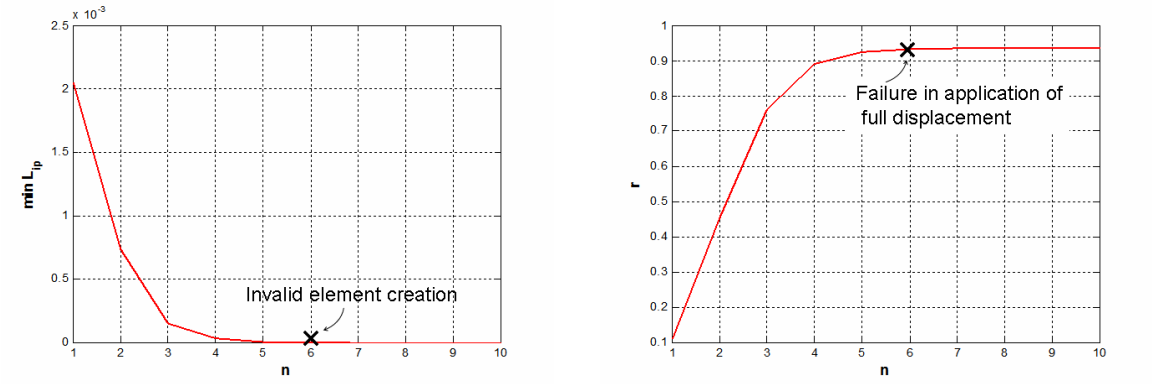
When the same simulation is repeated with the classical edge-spring method, the method fails almost immediately for such large amplitudes. The invalid elements are generated at the leading edge, where the mesh quality is crucial (Figure 40(a)). The next figure, Figure 41, also illustrates this failure in terms of quality measures. At the sixth time step of the simulation an invalid element is created such that $min_{L_{ip}}$ becomes zero; that is, one of the elements becomes exactly a triangle-quadrilateral which is an unacceptable grid cell from computational point of view. For a simulation with $N = 80$, the difference between the behaviors of the two methods (Figure 42) is recognized at the very beginning of the motion,

where the damage to the mesh is tremendous for the edge spring analogy (ES).



(a) Spring analogy (b) Ball-vertex method

Figure 40: (a) Invalid element generation at 6th step with the classical edge spring method (b) Valid mesh with the ball-vertex method at 500th step for an airfoil with 996 quads and 937 vertices



(a) (b)

Figure 41: Spring analogy: Quality plots for pitching and plunging airfoil, with 996 quads and 937 vertices.

Notice that we use fairly large imposed displacements per time step with the purpose of highlighting possible differences in the behavior of the two schemes in extreme cases. Indeed, as expected, by reducing the time step length and thereby reducing the magnitude of the imposed displacements to smaller values, both methods work. However, the ball-vertex method would be far superior for implicit flow solvers in which very large time steps and therefore large displacements are typical.

Next, we investigate robustness of the scheme, using the exact same problem with much finer mesh, namely 3,862 elements and 3,682 nodes, 300 of which are on the airfoil surface (Figure 43). As can be seen in Figures 44, 45, and 46 the behavior of the ball-vertex scheme

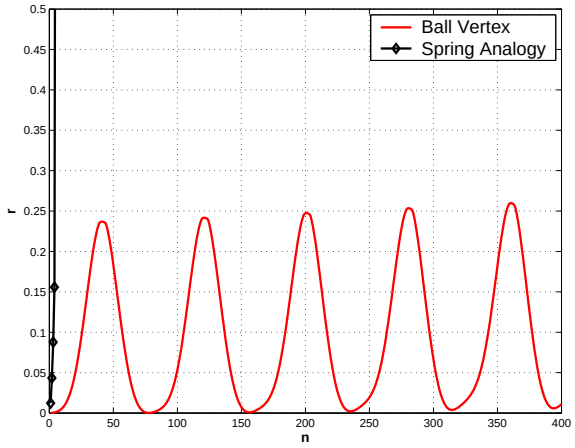


Figure 42: Quality comparison between the ball-vertex and the spring analogy.

does not vary for different mesh density and sizes. In both cases, one can observe a nicely periodic behavior indicative of a mesh deformation process, which demonstrates that the damage to the mesh as a result of the repetitive pitching and plunging motion is, in fact, minimal. With this example we observe that a certain robustness exists for the methodology in terms of the density and the size of the elements.

Lastly, when mesh deformer is coupled with a flow solver, both the coarse and the fine airfoils at higher angle of attacks might require further refinement to more accurately resolve vortex shedding over the surface and in the wake.

3.8.3 Pitching and Plunging Wing

Next, we consider the three-dimensional case of a pitching and plunging wing. A view of the wing, discretized with non-simplicial elements, is shown for the initial configuration in Figure 47, downstroke in Figure 48 and upstroke in Figure 49. A similar problem for a tetrahedral grid was studied in Reference [22]. The deformation problem is solved for four full cycles with N=100. For this example, we used a pitching amplitude of $\theta_0 = 15$ deg, and a plunging amplitude of $h_0 = 0.4$ chords and

$$\theta_n = \theta_0\Big(\frac{\pi}{180}\left(-\sin(2\pi n/N + 0.2\pi) + \sin(2.2\pi)\right)\Big), \tag{69a}$$

$$h_n = h_0 \sin(2\pi n/N). \tag{69b}$$

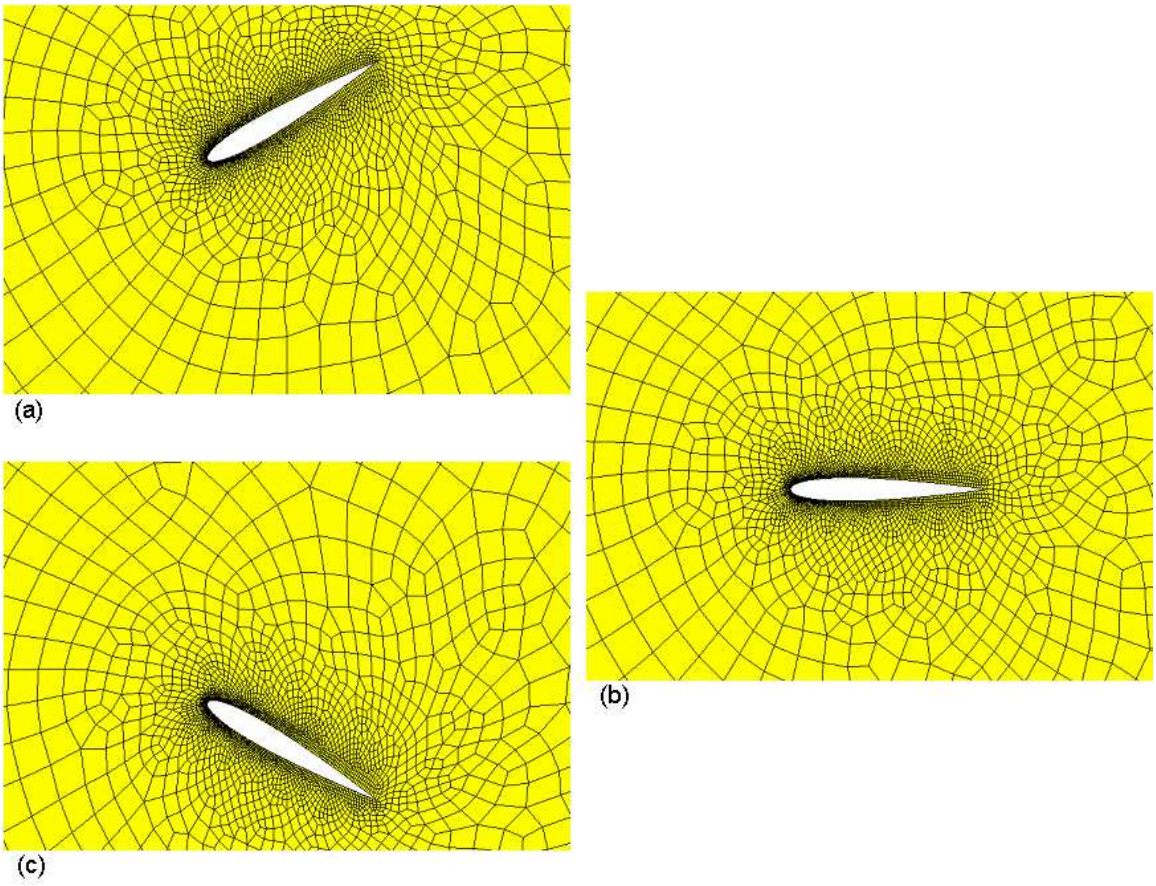


Figure 43: Pitching and plunging airfoil with 3,682 nodes; upstroke (a), initial configuration (b) and downstroke (c).

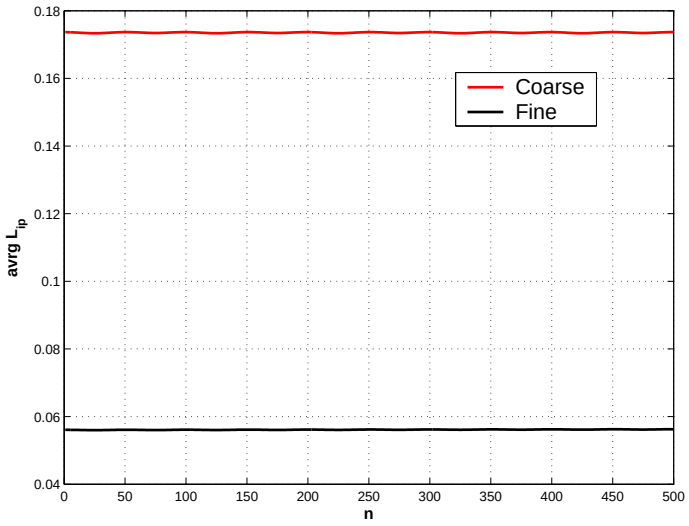


Figure 44: Average face spring length comparison of the ball-vertex method using fine and coarse grids.

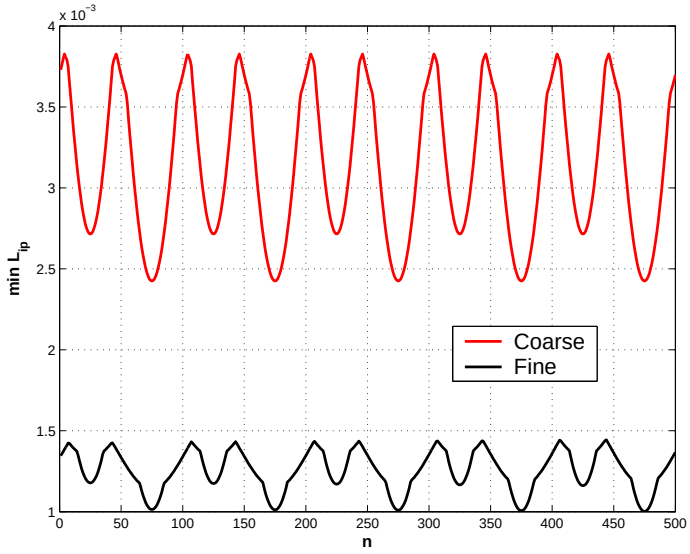


Figure 45: Minimum face spring length comparison of the ball-vertex method using fine and coarse grids.

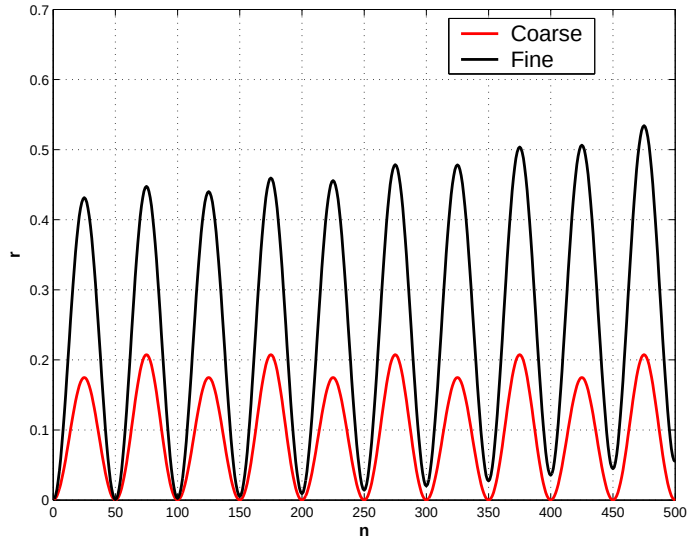


Figure 46: Compliance residual in the Frobenius comparison of the ball-vertex method using fine and coarse grids.

Again, for this problem, we quantify the quality of the grid by using the metric-based distortion measure r defined in equation (68), and the average and minimum ball radii vs. time. Specifically, Figure 50 (c) shows that while r grows negligibly small in the order of 10^{-2} at the end of the first cycle, $avrg_{L_{ip}}$ peaks at the up and down strokes with very mild variations from the initial configuration. When computing r for this problem, each hexahedron was split into five non-overlapping tetrahedra. The constant metric of each tetrahedron was computed, and finally, the values of the five tetrahedra were averaged to obtain an estimate of the metric of the hexahedron.

Figure 50 (b) shows the minimum length of the ball-vertex edges throughout the simulation, i.e., the minimum radius of the maximum ball-inscribed circle. The ordinates are reported in a logarithmic scale to better appreciate the behavior of the solution at the points of maximum distortion.

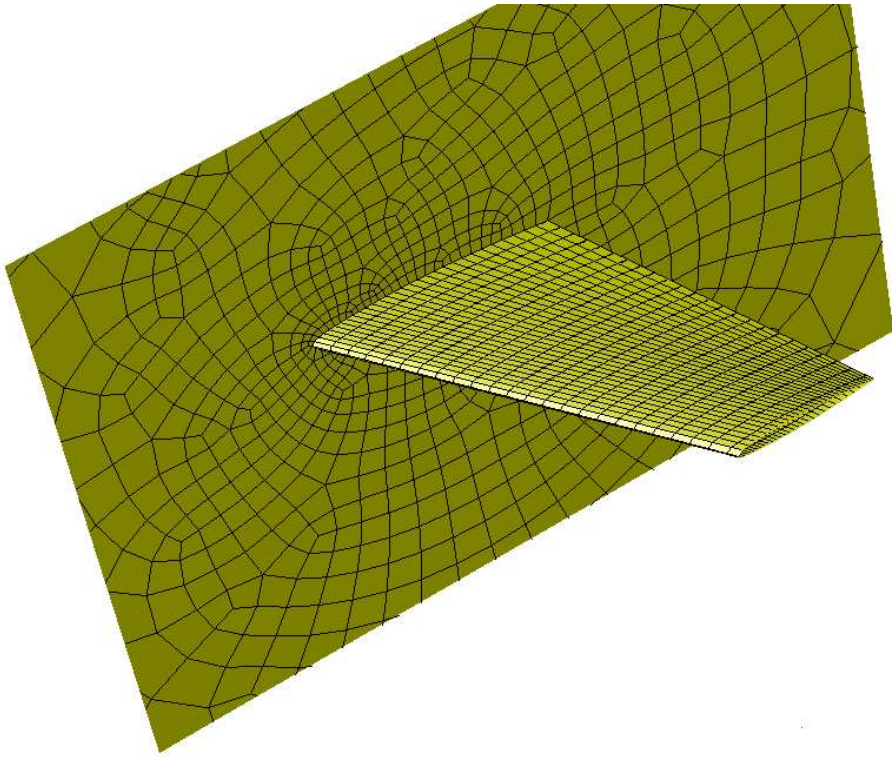


Figure 47: Pitching and plunging wing: initial mesh.

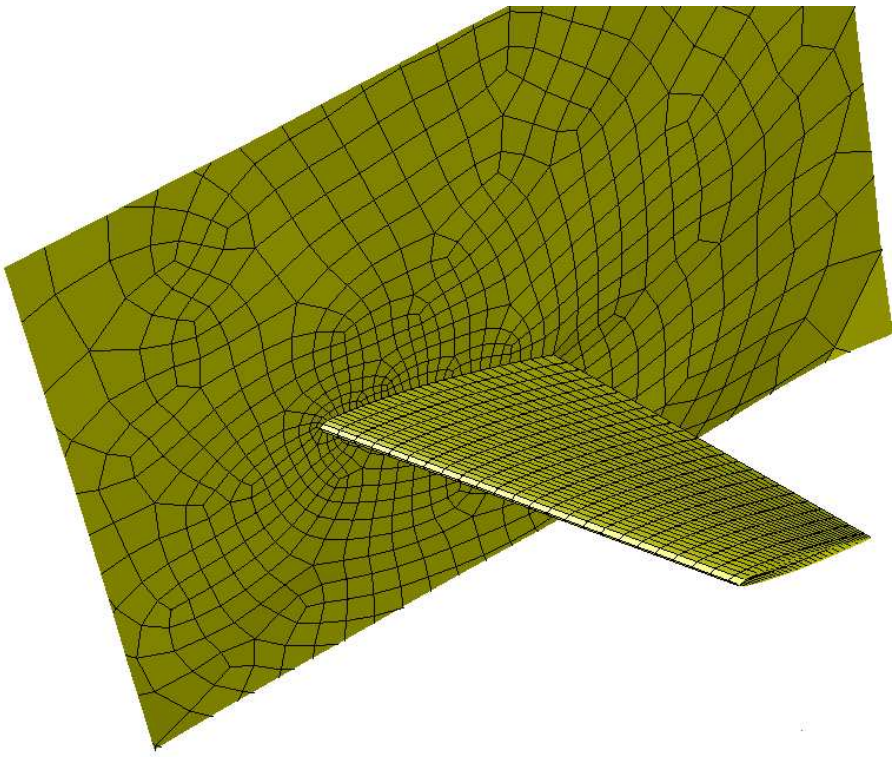


Figure 48: Pitching and plunging wing: down stroke.

3.8.4 Flapped Airfoil

The last example deals with an airfoil with a leading edge slat and a trailing edge flap that has been used as an effective high-lift device [37, 47, 78, 26, 86, 63, 30]. The triangular mesh contains 5,534 elements, and its initial configuration is shown in Figure 51(a). Both leading and trailing edge flaps were then moved towards their fully-stored configuration, shown in Figure 51(b). The flaps were subsequently deflected back again to their initial positions in Figure 51(c). This sequence of retraction-extraction of the flaps defines a full cycle of the simulation, which was here conducted using 300 incremental steps.

The sequence was repeated multiple times. While, Figure 51(d) shows the mesh after the fourth cycle, Figures 52 and 53 depict (a) the initial configuration with a zoom of the grid in the regions of the leading and trailing edges of the profile, and (d) the grid obtained at the completion of the fourth cycle. A visual inspection of the figures shows barely noticeable changes between the initial and final grids, even though the mesh has been subjected to

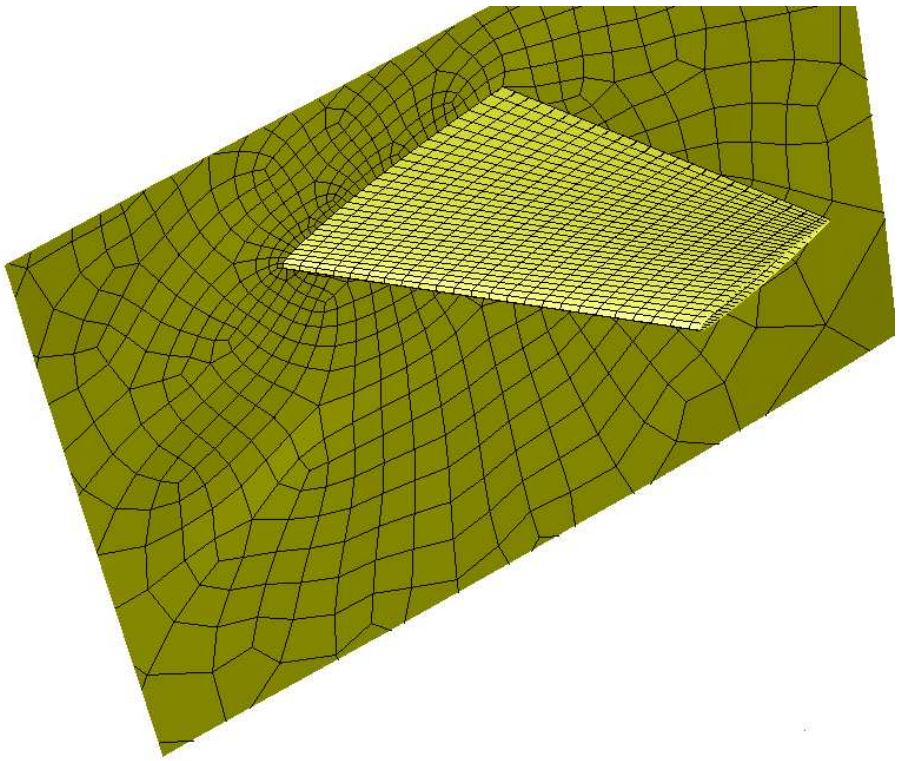


Figure 49: Pitching and plunging wing: up stroke.

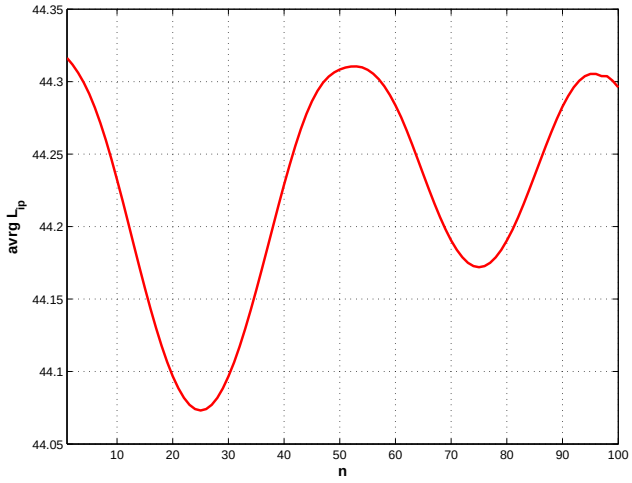
dramatic levels of compression and subsequent stretching in the slot areas.

A more quantitative measure of the performance of the method can be obtained by the plots of Figure 54 (a) and (b). Figure 54 (b) reports the distortion measure r vs. the time step. As for the previous examples, it appears that the method is able to almost recover the initial configuration, as already apparent from the visual inspection of the grid. However, the peak values of r corresponding to the stored flap configurations show growth with an increasing number of cycles.

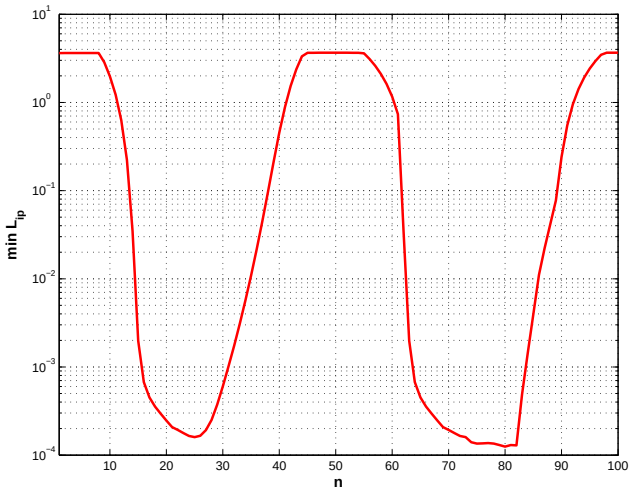
Similarly, the plot of Figure 54 (a), which illustrates the minimum ball radius vs. time, reveals a dramatic compression of the grid and evidences of some degradation with an increasing number of cycles.

3.8.5 Bending Wing

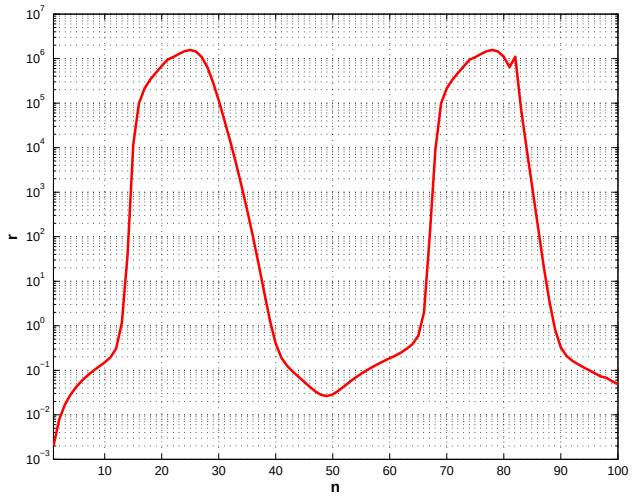
As our last example, we consider again the wing based on a ONERA D symmetric profile similar to the pitching and plunging case. The grid is constructed such that it has 21,432



(a) avrgLip



(b) minLip

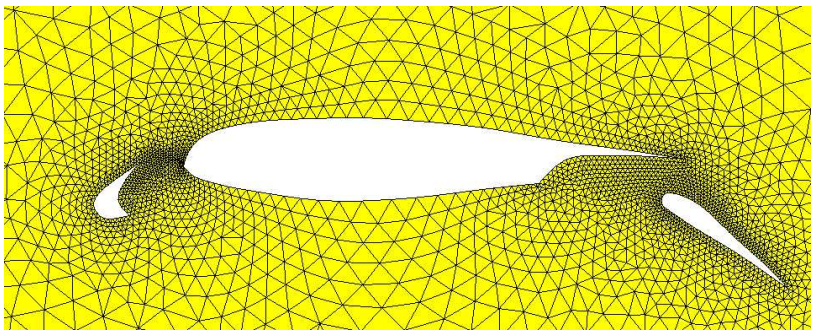


(c) compliance residual

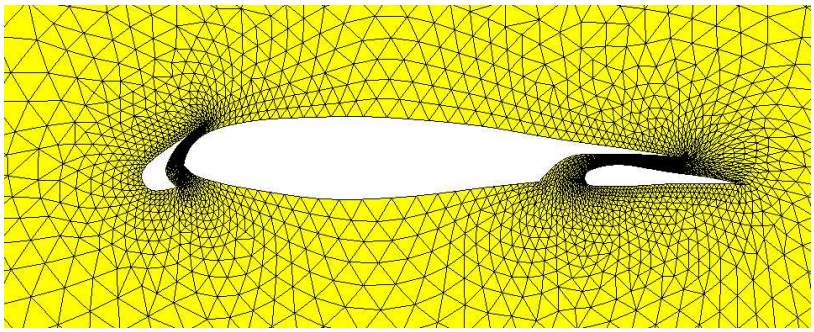
Figure 50: Pitching and plunging wing deformation quality.

nodal points, 1,550 of which are located on the wing surface itself. We assume the wing is clamped at its root and bent with a tip bending amplitude of a quarter span, i.e., 50 percent of the half span. The amplitude is zero at the root and increased linearly along the spanwise direction. The deformation problem is solved for four full cycles with N=100. Figure 55 shows the wing at the upstroke and down stroke positions. The side view in Figure 56 demonstrates the significance of the bending.

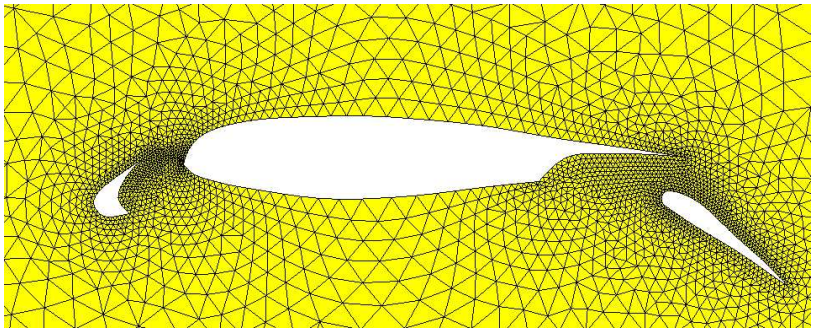
Even for this example, in which the wing is involved in extremely large displacements, the mesh quality has been successfully maintained. Different from the pitching and plunging wing, we don't have any rigid body motion. The mesh deteriorates at higher cycles as shown in Figures 57 (a), (b) and (c). However, the scale of deterioration is not dramatic. One can easily tolerate such deviation provided that the mesh motion doesn't fail, i.e., no degeneracy, which could be fatal during simulation, occurs.



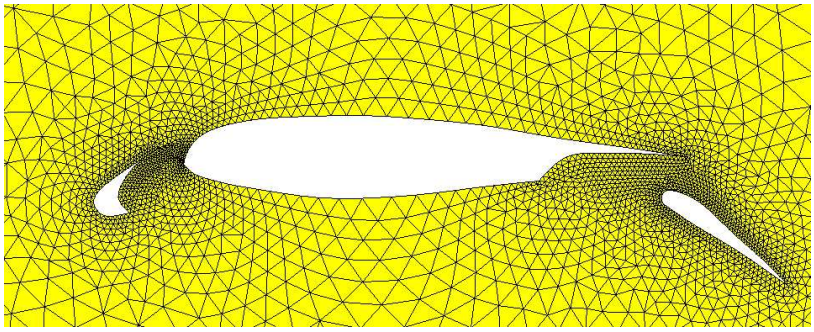
(a) initial



(b) half cycle

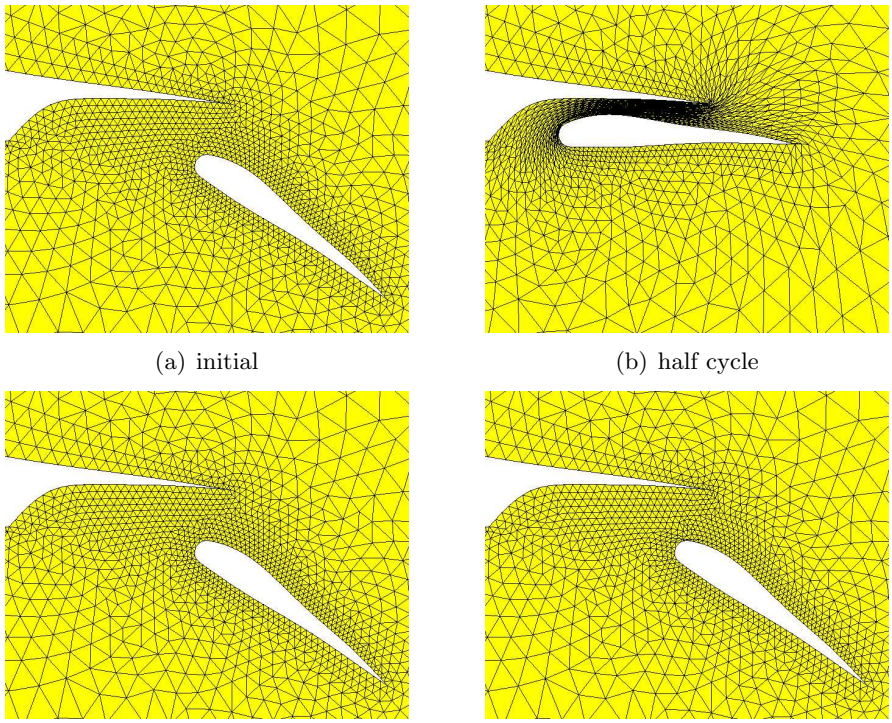


(c) end of first cycle



(d) end of fourth cycle

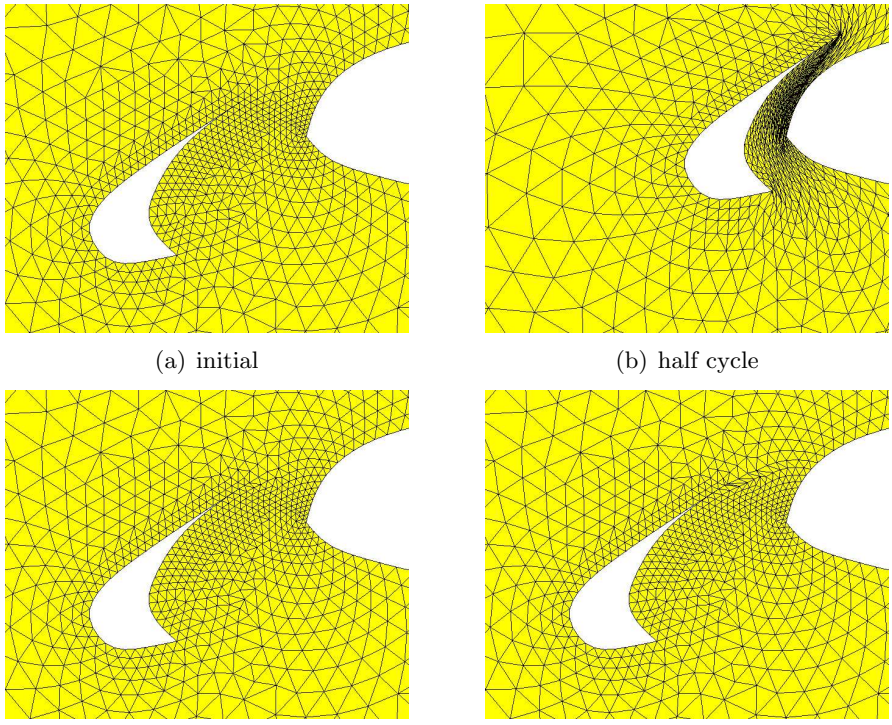
Figure 51: Mesh deformation of multi-element airfoil.



(a) initial (b) half cycle

(c) end of first cycle (d) end of fourth cycle

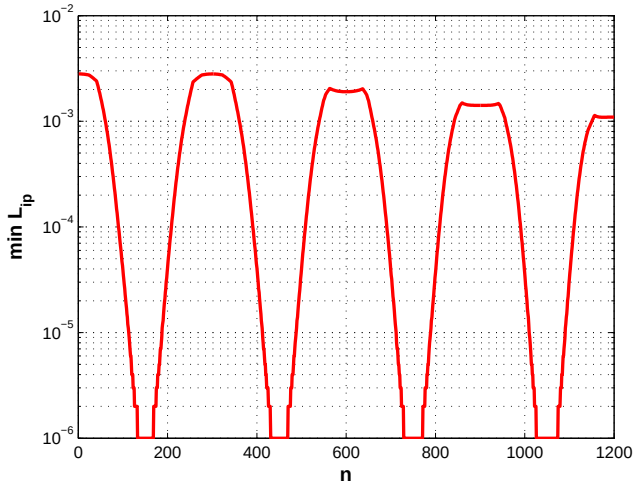
Figure 52: Mesh deformation around flap, detail of deforming multi-element airfoil.



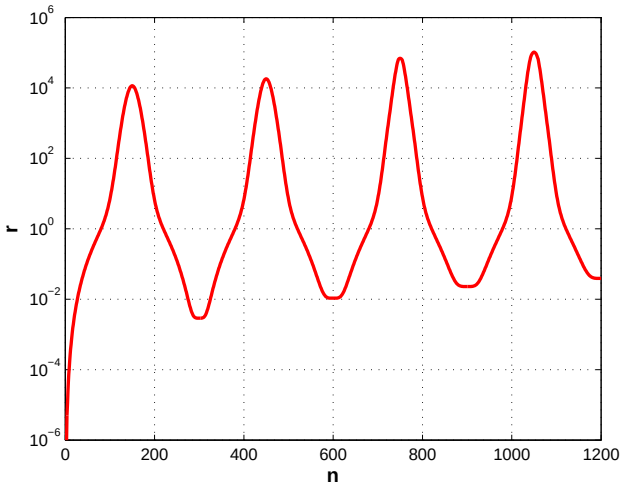
(a) initial (b) half cycle

(c) end of first cycle (d) end of fourth cycle

Figure 53: Mesh deformation around slat, detail of deforming multi-element airfoil.

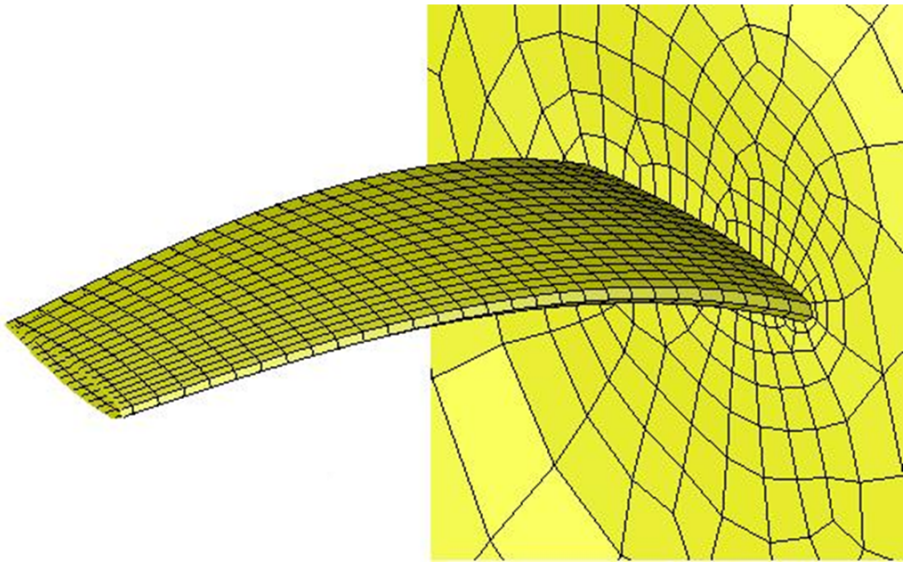


(a) min L_{ip}

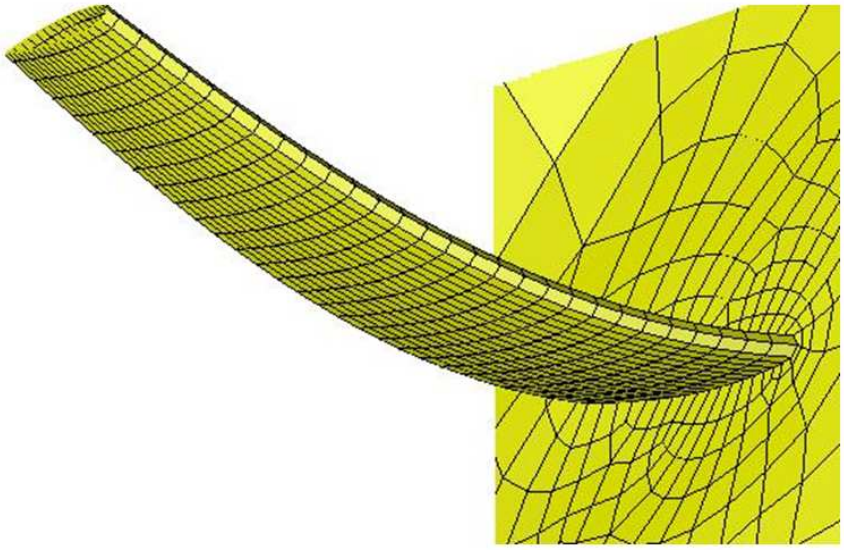


(b) Compliance residual r

Figure 54: Multi-element airfoil deformation quality.

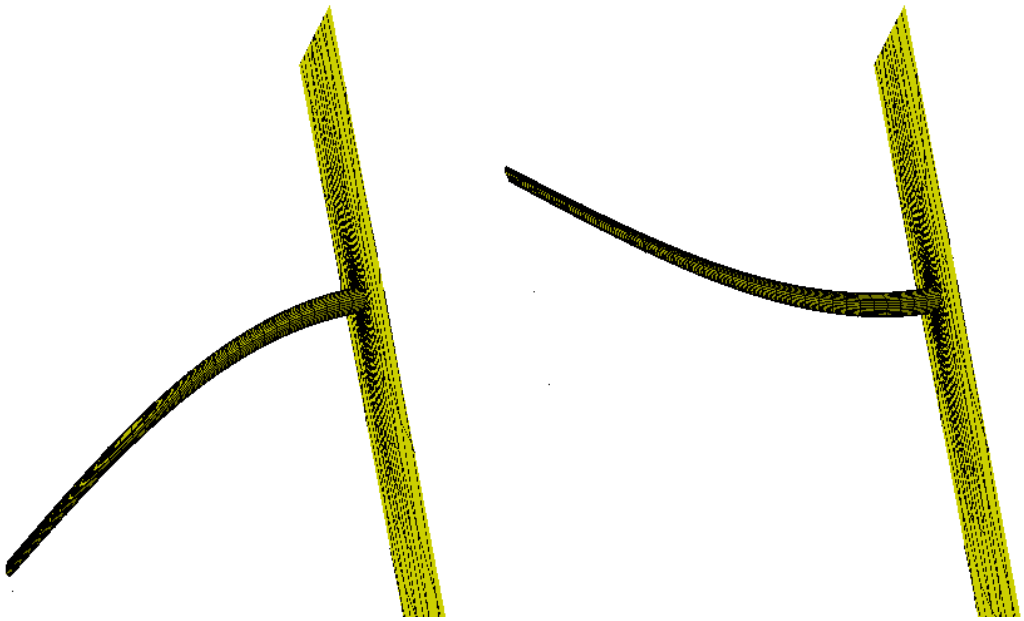


(a) Downstroke



(b) Upstroke

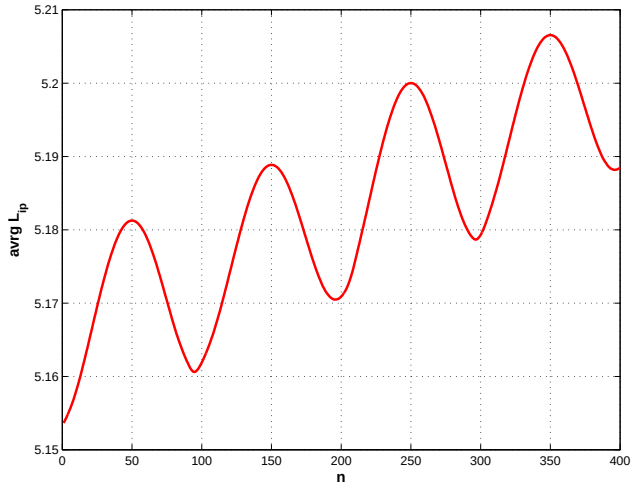
Figure 55: Bending wing



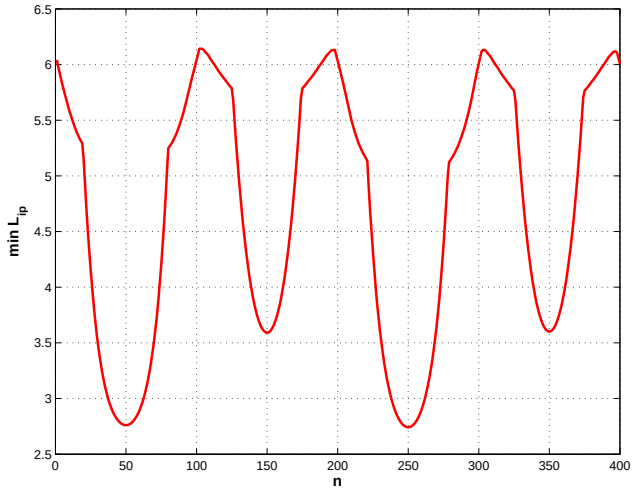
(a) Downstroke, side view

(b) Upstroke, side view

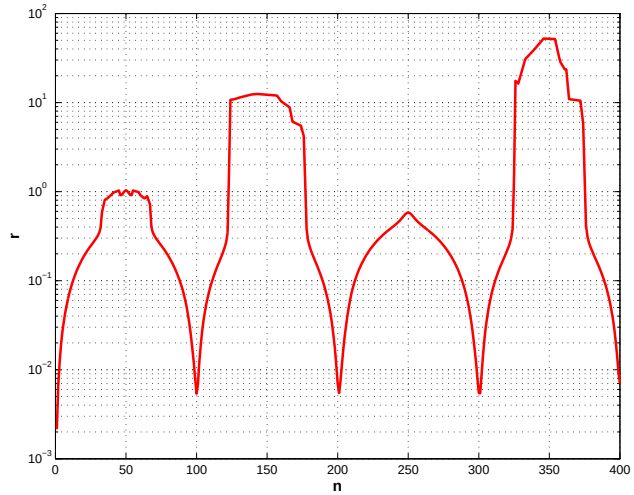
Figure 56: Bending wing, side view



(a) average L_ip



(b) minimum L_ip



(c) compliance residual r

Figure 57: Bending wing deformation quality.

Chapter IV

METRIC-BASED MESH DEFORMATION

4.1 Introduction

For problems with large spatial and temporal variations, in order to increase accuracy, and decrease mesh-dependency and computational cost of the numerical simulation, the use of adaptive methods is necessary. Adaptation is especially appreciated in computational fluid dynamics (CFD) problems, in which the solution depends on both the flow geometry and the flow conditions. That is, for the same geometry, the most appropriate mesh can significantly vary at different Mach and Reynolds numbers for steady flows. Obviously, care must be taken in the right clustering of the meshes since each particular flow entails a unique meshing. Furthermore, for unsteady problems, features of flow and/or geometry change continuously, resulting in frequent remeshing of the domain.

Usually, most CFD simulations are required to solve problems involving highly demanding physical features such as vortices, flow separation, and compression waves. These phenomena are characterized by regions of sharp gradients of flow variables embedded in regions of smooth variations. However, accurately pinpointing the locations of these regions may not be possible prior to simulation. Therefore, the only way to avoid intuitive and, for time-dependent flows, frequent meshing of the entire domain, is to take advantage of adaptation techniques.

Adaptivity can be achieved in several ways: h-, p- and r- methods. For h-adaptivity, the grid is refined or coarsened by adding or deleting node points through topological operators, while p-methods change the degree of the polynomial used to approximate each element. In case of r-adaptation, also known as the moving-mesh method, grid points are repositioned as needed while preserving initial connectivity and the total number of nodes. In principle, r-adaptation seeks the optimal mesh within any given topology. H- and p-adaptations are extensively used in studies as they are reliable for finite element solutions. However, in

practice, although r-adaptation techniques are not new among the adaptive methods, they are not used as often in numerical analyses, largely because of the difficulty in developing a general and robust mesh deformation technique that does not create degenerate mesh elements. This major drawback of r-adaptation, on the other hand, could be overcome by the ball-vertex methodology, which has proven to be a powerful deformation tool [3, 5]. R-methods have significant advantages such as relative ease of coding compared to topological alterations, and the ease of implementing the method into existing fixed mesh codes. Furthermore, r-adaptation is the only way of adapting for structured meshes if the connectivity as well as the total number of nodes of the mesh remains unchanged.

Often, with a known error distribution, r-adaptation could assure precise solutions for PDEs. Clustering nodes around high-gradient regions while coarsening them at low-gradient regions would enhance the solution considerably as in problems that involve shocks, boundary layers, and vortices. The error estimation could be determined in advance, a-priori, or could be as a result of a solution on the current mesh, a-posteriori. In the former case, the distributions of the size, shape, and orientation of the element (i.e., quad, hex, triangle, tetrahedron) for the final mesh are known. Based on this information, target metrics are computed over the domain and compared with those of the current mesh. Then, mesh is deformed accordingly by employing a deformation technique driven by a force field that is generated using an error estimator. In this study, we use the ball-vertex method due to its ability for large amplitude repositionings. Additionally, for a-posteriori error estimation, we use the Hessian recovery technique, where second derivatives of the solution determines nodal density, as explained in detail in section 2.3.3. Our focus here is exclusively on two-dimensional meshes, from which we obtained very promising results. Hence, we expect our dynamic mesh algorithm to be equally applicable to three-dimensional cases.

4.2 Literature Review

Over the past ten to twenty years, with the development of various error estimation techniques, mesh adaptation has become an efficient instrument for simulating many physical phenomena, including boundary layers, wakes, shock waves, and vortices.

In particular, h- and hr-adaptivity techniques have been more commonly studied. We will briefly discuss three main techniques commonly addressed in past studies: remeshing, subdivision, and local modification techniques. With regard to the first, Peraire et al. [60] suggested a directional approach in constructing anisotropic grids with resolution along rapidly changing error estimate directions. They used a two-dimensional hr-adaptive remeshing for inviscid steady state flows of triangular meshes. Density was the flow variable for which the error was estimated through Hessian recovery. Frey and Borouchaki [36, 19, 20] also applied adaptive remeshing in terms of the Delaunay Kernel and showed that remeshing, although powerful, can be inefficient for already refined meshes in which only a few elements needed to be refined. The second group, element subdivision methods, have been utilized in controlling the element shape and size with specific split orders. Liu et al. [50] and Rivara et al. [68] developed such subdivision techniques using bisectioning, which adjusts each tetrahedron based on a shape function. The techniques used in element subdivision result in meshes that sometimes tend to be over refined, and coarsening beyond the initial mesh is not possible because the coarsening algorithm is based on the undoing of the refinements. The third group, local modification techniques, apply local mesh modification operators in a certain order. Among these operators, edge/face swapping, edge/face/region splitting, edge collapse, and vertex movement are the currently used primitives. Buscaglio et al. [23], Castro-Diaz et al. [27], Habashi et al. [38], Yahia et al. [89], Dompierre [32], Pain et al. [58], Remaki et al. [67] Eduardo [33], Webster et al. [88], Yannis [90] and Tam et al. [79] made use of these local mesh modifications for adaptation purposes. For instance, in references [38, 89, 32], which are three-part series of an anisotropic adaptation, the authors discuss applications of topological alterations to structured and unstructured meshes. They simulate mainly hypersonic and transonic flows with use of metric-driven error estimators. In addition to topological modifiers, they have a spring analogy-based Laplacian smoothing of the nodes, which has to be modified with additional energy terms in order to compensate for the failure of spring analogy typical in large displacements. Again, Pain et al. [58] used metrics, described by the Hessian of the solution domain, as a means of error measurement. They basically superposed metrics of

different flow variables for steady and transient finite element computations. As an alternative approach, Remaki [67] showed two-dimensional applications of local h-adaptation coupled with shock filtering techniques in order to resolve weaker shocks more precisely on triangular grids.

R-adaptation or moving mesh methods were studied as early as 1981 by Miller et al. [55] but improved in 1998 by Carlson and Miller in 1998 [25]. Their idea was to move the relocated mesh points by minimizing the residuals and using the gradient-weighted finite element method to adapt the current mesh. While the method proves effective for problems with sharp-moving fronts, the crucial issue is that it is not intended for smooth solutions, turbulent chaotic solutions, or problems in purely conservative forms.

In 1988, Field [29] described Laplacian smoothing, in which a vertex is repositioned according to some weighted sum of position vectors of its edge-connected vertices. Even though this method tends to produce isotropic meshes, it is often employed as part of the most current h-adaptation schemes. In a similar smoothing attempt, Bank [11], in 1997, developed a new algorithm based on a-posteriori error estimation in which node positions are updated as a result of the local minimizations of a particular shape function. Likewise, Bern et al. [7] worked on optimization-based smoothings, compared them for unstructured grids using quality criteria such as angle, area, aspect ratio, and inscribed radius, and showed that they outperform Laplacian type smoothing.

Another common methodology in r-type adaptations is moving mesh PDEs, where a set of partial differential equations are derived from a known error estimation. One of the early examples of moving PDEs was introduced by Zegeling et al. [92]. In brief, they described an adaptive moving mesh method based on a coordinate transformation between physical and computational coordinates. The transformation was basically the solution of the adaptive mesh partial differential equations derived from minimizing of a mesh-energy integral. Their strategy emphasized the decoupling of the adaptive mesh PDEs from the physical PDEs of the problem. Later, Zegeling [91] compared this method with the moving finite differences method based on an equi-distribution principle with smoothing in both spatial and temporal directions.

Different from Zegeling et al., Cao et al. [24] integrated the moving mesh PDEs with the physical PDEs of the problem. Specifically, their r-adaptive finite element method utilizes a parabolic moving-mesh PDE that is formed based on an error estimator with spacial discretization. Another example of importance in continuous PDE approach was developed by Liao [17, 73], who distributed the grid nodes according to a given analytic or discrete weight function of the spatial and time variables. Specifically, the PDE for the mapping function, written in terms of weighting functions, is solved with the equi-distribution condition in one-, two-, and three-dimensional regions. In a successive study, Baines [10] used a least squares minimization technique in which node displacements are calculated as a part of the unknowns. The primary contribution of this work was to improve the results of the descent least squares method near discontinuities, e.g., shocks, by moving grid points around high variation regions.

Several recent studies [62, 49] pursue a novel strategy in which an r-adaptation is followed by mesh coarsenings and refinements, or vice versa. For example, Popiolek [62] applied h-refinement followed by nodal repositioning with first-order derivative error indicators. In a similar effort, Lin et al [49] coarsened deformed meshes in order to remove distorted elements and then further refined the mesh.

Error estimator studies compose another key part of the r-adaptation literature (or adaptation literature in general) because adaptation in its current form is made possible only after the introduction of error estimators. An error estimator can simply be defined as a measure of how the mesh should be modified. In other words, it provides information on desired nodal density and mesh alignment. Error estimation can be categorized into two basic methods: error residual methods and recovery methods. Error residual methods were first developed for linear elliptic problems by Babuska et al. [9] and later extended to Navier-Stokes equations by Oden et al. [64]. The latter study proposed two residuals due to spatial discretization, one derived from the momentum equation and the other from the continuity equation. One fundamental disadvantage of residual methods is that they can indicate the location of an error but not its magnitude. For example, Marcum et al. [54], Walter et al. [87], and Scalabrin et al. [71] used such error definitions with first-order derivatives,

called “error sensors”, in order to adapt isotropic triangulations. Recovery methods, or ZZ-estimators, introduced by Zienkiewicz et al. [93, 18] can detect the magnitude of an error. For these types of methods, the approximate solution is often post-processed in order to get an enhanced and presumably more accurate solution.

4.3 Revisited Ball-Vertex Method

An elaborate explanation of the ball-vertex methodology with numerous example problems has been given in Chapter 3. Before proceeding into the metric-based r-adaptation, we will briefly review the ball-vertex technique here.

The method is formulated so as to avoid, by construction, the generation of invalid elements even for very large boundary displacements. We showed that in all cases, simplicial and non-simplicial, invalid elements are generated by the same collapse mechanism. Specifically, an invalid element is formed when a mesh vertex leaves its ball, i.e. the polyhedral cavity formed by all triangular faces (in three dimensions) or edges (in two dimensions) that are edge-connected to the vertex. We also showed that, in all cases above, collapse of elements is avoided by connecting each vertex in the grid with its normal projection on the ball boundaries via linear springs.

These springs effectively constrain each vertex within the polyhedral ball that encloses it, contrasting the possible collapse mechanisms of the grid elements. Therefore, this method avoids mesh entanglement during deformation by explicitly enforcing the vertex containment condition. Numerical examples have shown that the method behaves well, and is able to guarantee good quality meshes even after repeated cycles of very large amplitude deformations.

4.4 Adapting the Mesh with a Force Field

For mesh deformation problems, one usually knows the boundary displacements, and needs to move the internal and sliding nodes accordingly. However, in a more general definition of adaptation, movement of the internal nodes may or may not be driven by a boundary motion, for example mesh points can be relocated in order to transform a current mesh into a desired target mesh.

In this work, we adapt the mesh by relocating vertices using the previously explained ball-vertex methodology. However, different from the pure ball-vertex technique, compression and elongation of the springs are enabled by a pre-defined external force field instead of a pre-defined boundary movement. Throughout this process the topology remains fixed because both the refinement and the coarsening of the elements are prevented. The force field computed using an error estimator is composed of individual "resultant" force vectors, each acting on a single vertex. Figure 58 illustrates such a force distribution for a structured two-dimensional grid. Here, clustering is done towards the left bottom corner of the domain, as indicated by the direction of the applied forces. The vertices of initial uniform mesh are compelled to move, by the force field, at a distance proportional to the magnitude of the corresponding force vectors. The displacements of the nodal points, are calculated by establishing a force equilibrium throughout the mesh domain, which is explained below in detail. This equilibrium condition leads to a sparse set of linear equations for the mesh field, which is eventually solved by the minimum residual method.

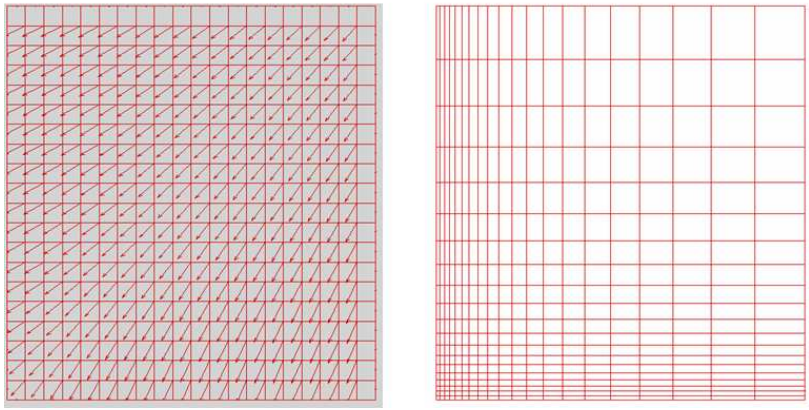


Figure 58: Vertex repositioning of a quad mesh with a metric-based force field. Left: an initial mesh and an initial force field, Right: a final mesh.

4.4.1 Computation of the Force Field

Originally, we are given an initial mesh and a target metric $\bar{M}_K$, and using this information we aim to derive the force field. Let us begin the computations with the calculation of the force at vertex i due to the stretching of edge ij (Figure 59 (left)). Let $\boldsymbol{e}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$ represent the edge vector joining vertex i to vertex j, and vector $\tilde{\boldsymbol{e}}_{ij}$ represent the target

edge vector that coincides with $\boldsymbol{e}_{ij}$ but has a unit length in control space and a desired edge length prescribed by $\bar{M}_K$ in physical space.

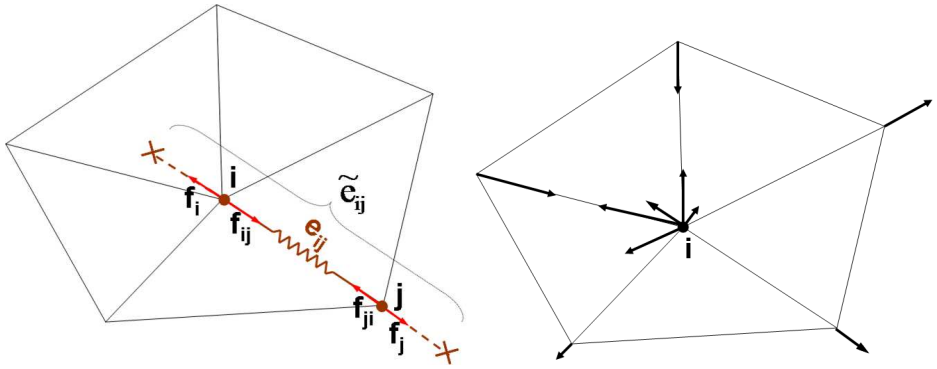


Figure 59: Left: Spring force due to the stretching of edge $\boldsymbol{e}_{ij}$. Right: Individual edge-spring forces acting on node i

$$\tilde{\boldsymbol{e}}_{\boldsymbol{ij}} = \frac{\boldsymbol{e}_{ij}}{\sqrt{\boldsymbol{e}_{ij}\bar{\boldsymbol{M}}_{\boldsymbol{K}}\boldsymbol{e}_{ij}}} \tag{70}$$

Our primary goal here is to stretch the edge $\boldsymbol{e}_{ij}$ and bring it up to the same edge length as $\tilde{\boldsymbol{e}}_{ij}$. We achieve this by applying an appropriate external forces $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ on the bounding nodes i and j. In order to calculate the external forces we first investigate the internal spring force. The spring force is by definition proportional to the stretching and the stiffness constant. So we have

$$\boldsymbol{f}_{ij} = k_{ij}(\boldsymbol{u}_j - \boldsymbol{u}_i)\boldsymbol{i}_{ij}\boldsymbol{i}_{ij} = -f_{ji} \tag{71}$$

where k_{ij}, $\boldsymbol{u}_j$, $\boldsymbol{u}_i$, and $\boldsymbol{i}_{ij}$ stand for spring stiffness, the displacement vector at j, the displacement vector at i, and the unit vector along edge ij, respectively. Notice that the external forces acting on the nodes i and j are $\boldsymbol{f}_i^{ij} = -\boldsymbol{f}_{ij}$ and $\boldsymbol{f}_j^{ij} = \boldsymbol{f}_{ij}$ respectively. The stiffness constant in equation (71) is trivial to compute, i.e., $L_{ij} = 1/k_{ij}$, however we need to find a proper way to express the required stretching in terms of the known quantities: current mesh geometry and Riemaniann target metric. For that purpose we use the vector difference between $\tilde{\boldsymbol{e}}_{ij}$ and $\boldsymbol{e}_{ij}$, that is

$$(\boldsymbol{u}_j - \boldsymbol{u}_i)\boldsymbol{i}_{ij}\boldsymbol{i}_{ij} = \tilde{\boldsymbol{e}}_{ij} - \boldsymbol{e}_{ij} \tag{72}$$

We plug the definition of target edge $\tilde{\boldsymbol{e}}_{ij}$ (eqn. 70) into the stretching expression:

$$(\boldsymbol{u}_j - \boldsymbol{u}_i)\boldsymbol{i}_{ij}\boldsymbol{i}_{ij} = \frac{\boldsymbol{e}_{ij}}{\sqrt{\boldsymbol{e}_{ij}\bar{M}_K\boldsymbol{e}_{ij}}} - \boldsymbol{e}_{ij} = (\frac{1}{\sqrt{\boldsymbol{e}_{ij}\bar{M}_K\boldsymbol{e}_{ij}}} - 1)\boldsymbol{e}_{ij} \tag{73}$$

Hence, the individual external force at node i can be written using equation 73 as

$$\boldsymbol{f}_i{}^{ij} = k_{ij}(1 - \frac{1}{\sqrt{\boldsymbol{e}_{ij}\bar{M}_K\boldsymbol{e}_{ij}}})\boldsymbol{e}_{ij} = -\boldsymbol{f}_j{}^{ij} \tag{74}$$

Figure 59 (right) shows these individual force contributions from all the edges.

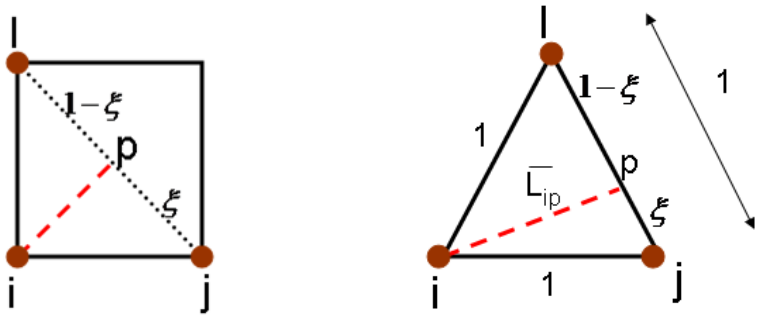
Finally, in order to get the resultant net force acting on node i, corresponding external forces $\boldsymbol{f}_i{}^{ij}$ and $\boldsymbol{f}_i{}^{ip}$ for each edge and each face-vertex spring respectively, are calculated, and summed up. However, because face-vertex springs have a non-unit length in metric space as illustrated by Figure 60 (b), the force calculation for those springs (i.e., equation (74) for $\boldsymbol{f}_i{}^{ip}$) has to be modified. Here, Figure 60 shows that the target triangle ijl is represented by a equilateral triangle in metric space, where the face-vertex spring clearly has a non-unit length, $\overline{L_{ip}} < 1$. We calculate the length of face-vertex in metric space from simple geometry as

$$\overline{L_{ip}} = \sqrt{\frac{3}{4} + (\frac{1}{2} - \xi)^2} \tag{75}$$

where ξ is again the interpolation coefficient as described in section 3.7 and $\overline{L_{ip}}$ is the target face-vertex spring length in Riemaniann metric space. Then, the force acting on the face-vertex spring ip becomes

$$\boldsymbol{f}_i{}^{ip} = k_{ip}(1 - \frac{\overline{L_{ip}}}{\sqrt{\boldsymbol{e}_{ip}\bar{M}_K\boldsymbol{e}_{ip}}})\boldsymbol{e}_{ip} \tag{76}$$

While $\boldsymbol{f}_i{}^{ip}$ appears to be controlling the location of the grid point i only, because p is a non-existent virtual point, it inherently controls positions of nodes j and l. This control is performed through the coefficient matrix calculations within the ball-vertex methodology and the transfer of $\boldsymbol{f}_i{}^{ip}$ to the neighboring nodes as in equation (64). Furthermore, it is important to note that one can compute the force field, only if a target metric distribution, $\bar{M}_K$, (i.e. an error estimation), is provided. This is because, it is the job of the error estimator to supply information about the target edge lengths. Depending on the nature of



(a) Target element in physical space

(b) Target element in metric space

Figure 60: Virtual spring edge length in metric space

the error estimation, the target metric distribution is either continuous or discrete. More often than not, a-priori definitions are continuous while a-posteriori distributions are discrete because the latter is usually obtained as a result of a numerical solution of a problem and requires a projection onto a background mesh. We will give examples for both types of error definitions in the section of numerical examples, Section 4.5.

4.4.2 Equations of Metric Driven Mesh Deformation

Now that the force calculations are complete, we can sum the force contributions from each edge and face-vertex spring connected to the same node to produce one final resultant force at each existing vertex. Once all the resultant forces are computed throughout the domain, a linear system of equations is formed:

$$K\boldsymbol{u} = \boldsymbol{f} \quad \text{subject to} \quad A\boldsymbol{u} = \boldsymbol{0} \tag{77}$$

The constraints $A\boldsymbol{u} = \boldsymbol{0}$ account for sliding node conditions: $\boldsymbol{n_i} \cdot \boldsymbol{u_i} = \boldsymbol{0}$, where $\boldsymbol{n_i}$ is the normal to the boundary at vertex i. The constraint matrix A is a rectangular matrix that contains unit normal vectors at the sliding nodes. So, it constitutes only -1, 0 and 1 at its entries, and has as many rows as the number of constraint equations.

Certainly, the moving boundary conditions still apply through ball-vertex deformation. However, for most adaptation problems, one usually has three sets of nodes, a) stationary, b) sliding, and c) free nodes, which excludes the moving boundaries. Thus, in these type of problems, coefficient matrix K of equation (77) is composed of individual block matrices of

only the free and sliding nodes.

Sliding nodes, in particular, require special attention, because their movement is restricted to a specific direction. Although, the condition $A\boldsymbol{u} = \mathbf{0}$ enforces this restriction, the resulting forces on the sliding nodes should also be adjusted so as to agree with the constraints. Therefore, for edges that have one node on a sliding surface, the perpendicular force component i.e., force along the normal direction to boundary, is neglected. This particular force evaluation at the sliding nodes, is repeated for the face-vertex springs as well, in which case either node i or virtual point p is positioned on the sliding boundaries.

Before solving the current constrained equation system (eqn. 77) for the displacements, we transform it into an unconstrained set. For that purpose, we integrate the constraint coefficient matrix, A, with the coefficient matrix K into a "global" coefficient matrix through Lagrange multiplier method. The procedure is as follows. First, we start by writing the principle of virtual work, which states that the energy variation of the entire system must be equal to zero at equilibrium. In other words, the variation in strain energy must cancel out the variation in work done by the applied forces when the system reaches equilibrium.

$$\delta(\frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}K\boldsymbol{u}) - \delta(\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f}) = 0 \quad \textit{at equilibrium.} \tag{78}$$

where $\frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}K\boldsymbol{u}$ and $\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f}$ denote strain energy and the work done by the applied forces, respectively [74]. Next, we modify equilibrium criterion by adding the constraint condition, which does not affect the outcome in a practical way since $A\boldsymbol{u}$ is a constant and already set to zero. Thus,

$$\delta(\frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}K\boldsymbol{u}) - \delta(\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f}) + \delta(\boldsymbol{\lambda}^{\boldsymbol{T}}A\boldsymbol{u}) = 0 \tag{79}$$

$\boldsymbol{\lambda}$ represents the vector of Lagrange multipliers, which from virtual work point of view are the auxiliary mathematical variables that can be related to the constraint forces of the spring system [15]. $\boldsymbol{\lambda}$ has as many elements as the number of constraint equations, and together with the unknown displacements $\boldsymbol{u}$, it forms the unconstrained variables of the system. In particular, $\boldsymbol{\lambda}$ helps manipulate the equilibrium relation into

$$\delta\boldsymbol{u}^{\boldsymbol{T}}K\boldsymbol{u} - \delta\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f} + \delta\boldsymbol{\lambda}^{\boldsymbol{T}}A\boldsymbol{u} + \boldsymbol{\lambda}^{\boldsymbol{T}}A\delta\boldsymbol{u} = 0 \tag{80}$$

Since $\delta \boldsymbol{u}$ and $\delta \boldsymbol{\lambda}$ are arbitrary and linearly independent, the following equations are obtained by setting their coefficients to zero

$$\boldsymbol{K}\boldsymbol{u} - \boldsymbol{f} + \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{\lambda} = \boldsymbol{0} \tag{81}$$

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{0} \tag{82}$$

which, in matrix form, is

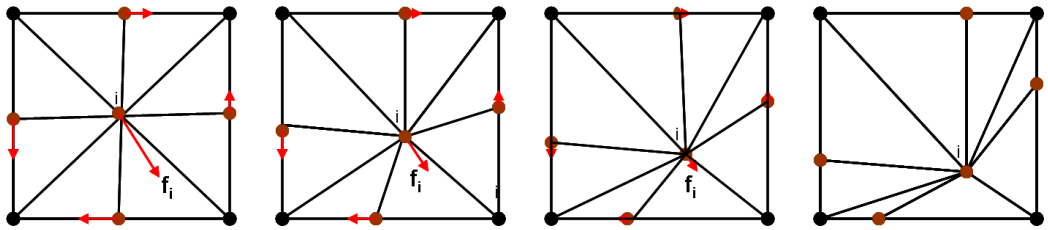
$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{A}^{\boldsymbol{T}} \\ \boldsymbol{A} & \boldsymbol{0} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix} \tag{83}$$

where $\boldsymbol{\lambda}$ is computed as part of the vector of unknowns although it does not have a direct contribution to the vertex displacements.

In the presence of constraints, the coefficient matrix of equation (83) is no longer positive definite, so we need to switch from the preconditioned conjugate gradients method, which was employed for the pure ball-vertex application, to minimum residual method (minres) [57]. The minimum residual method is a suitable algorithm for large, sparse, indefinite, and symmetric matrices since it requires neither positive definite nor symmetric coefficient matrices, despite the fact that it preconditions with a positive definite matrix. Besides, the solution requires fewer iterations since the main focus of the minres algorithm is on minimizing the residual.

During each iteration of the adaptation process, the equation system (83) is solved, and the node positions are updated accordingly. Needless to say, it will require more than one iteration for the initial mesh to completely comply with the target mesh. This is mainly because the resulting forces at the nodes are different than the individual force contributions by the single edges. Likewise the stretchings associated with the resultant forces, will be different than the stretching requirements associated with the individual edges. However, after each iteration, the gap between the current and the target mesh will get narrower and eventually vanish. Consecutive iterations begin with computation of a new force field based on the updated positions. This new force field will have smaller magnitudes compared to the previous iterations as shown in Figure 61, since the previous resultant forces moved the nodes to optimal locations so that connected edges are closer to their optimum

lengths. The iterations are stopped once the magnitude of the maximum force is equal to zero or to a certain user-defined limit, or when the average metric edge-based error residual, f_K (Section 2.3.4.1), converges. Clearly, it might be possible that the force field never diminishes completely, since point insertion is prevented in the current adaptation technique.



(a) Initial (b) First step (c) Second step (d) Third step

Figure 61: Vertex repositioning of a triangular mesh with a metric-based force field.

4.4.3 Simple One-Dimensional Example Problem

For the sake of simplicity, we explain the method in an illustrative one-dimensional problem. The current mesh in Figure 62 has two edges with lengths $h_{12} = 3$ and $h_{23} = 1$. Our goal is to generate a force field such that the final edge sizes will be uniform and equal to 2. That is, *node* 2 must move towards the left by one unit-distance. For this problem, an isotropic target metric is defined as

$$\overline{M} = \frac{1}{(\bar{h})^2} \tag{84}$$

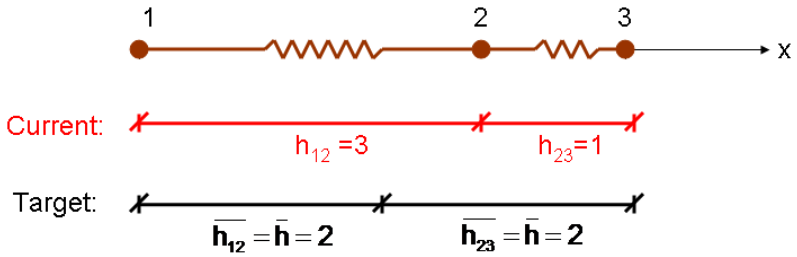


Figure 62: One-dimensional mesh deformation by an applied force.

Nodes 1 and 3, located at the boundaries need to be stationary i.e., $u_1 = 0$ and $u_3 = 0$; therefore, the net force on these nodes is zero. The net force acting on *node* 2, however, is

nonzero and has two contributions from *edge* 1 and *edge* 2 as follows (eqn. 74):

$$f_1^{12} = -f_2^{12} = \frac{1}{h_{12}} \left(1 - \frac{1}{\sqrt{\frac{{h_{12}}^2}{\bar{h}^2}}} \right) h_{12} = 1 - \frac{\bar{h}}{h_{12}} \tag{85}$$

$$f_2^{23} = -f_3^{23} = \frac{1}{h_{23}} \left(1 - \frac{1}{\sqrt{\frac{{h_{23}}^2}{\bar{h}^2}}} \right) h_{23} = 1 - \frac{\bar{h}}{h_{23}} \tag{86}$$

Here, superscripts 1 and 2 refer to *edge* 1 and *edge* 2, respectively. Resultant force f_2 is simply the summation of forces f_2^{12} and f_2^{23},

$$(k_{12} + k_{23})u_2 = f_2 = f_2^{12} + f_2^{23} \tag{87}$$

where k_{12} and k_{23} are the spring stiffness constants, which are inversely proportional to the edge lengths as described in previous chapters.

$$\left(\frac{1}{h_{12}} + \frac{1}{h_{23}} \right) u_2 = -1 + \frac{\bar{h}}{h_{12}} + 1 - \frac{\bar{h}}{h_{23}} \tag{88}$$

$$u_2 = \left(\frac{h_{12}h_{23}}{h_{12} + h_{23}} \right) \left(\frac{h_{23} - h_{12}}{h_{12}h_{23}} \right) \bar{h} = \bar{h} \left(\frac{h_{23} - h_{12}}{h_{12} + h_{23}} \right) \tag{89}$$

When the values of h_{12}, h_{23}, and $\bar{h}$ are plugged into equation (89), we get

$$u_2 = 2 \left(\frac{1 - 3}{1 + 3} \right) = -1 \tag{90}$$

which will result in a new position for *node* 2: $x_2 = x_2 + u_2 = 3 + (-1) = 2$. Then new edge lengths are

$$(h_{12})^{new} = x_2 - x_1 = 2 - 0 = 2 \tag{91}$$

$$(h_{23})^{new} = x_3 - x_2 = 4 - 2 = 2 \tag{92}$$

No further iteration is required for this problem because if we were to calculate a new force field, or f_2 in this case, it would be zero, the same as the residual. A quick check of equations (85) and (86) reveal

$$\begin{aligned} f_2^{12} &= -1 + \frac{2}{2} = 0 \\ f_2^{23} &= 1 - \frac{2}{2} = 0 \\ f_2 &= f_2^{12} + f_2^{23} = 0 \end{aligned}$$

Since the force field has already converged to zero; no further iterations are required.

4.5 Numerical Examples

4.5.1 Vertex Repositioning with A-priori Defined Metrics

Metrics for target meshes can be expressed through a-priori and a-posteriori error estimations. In this section, we provide examples in the context of the former. The mesh to be adapted is an arbitrary ten-by-ten square domain initially isotropically triangulated with 1919 nodes and 3676 triangles. The purpose is to get clustered mesh elements around $x = -2.5$, $x = 3.0$, $y = -2.5$ and $y = 3.0$ along the x and y directions. The average length of the initial isotropic elements is approximately 0.5; the target sizes of the mesh elements are given in equations (93) and (94), which define a linear and a exponential distribution, respectively. The clusterings are aligned with the principle axes, so the target metric is composed of only a diagonal component i.e., $\bar{M} = \begin{bmatrix} \frac{1}{h_x^2} & 0; & 0 & \frac{1}{h_y^2} \end{bmatrix}$.

Figures 63 and 66 show the initial and final meshes which are produced after 15 to 20 iterations while Figures 64, 65, and 67 exhibit the compliance histograms of the adapted meshes. The residual distribution in the figures refers to the metric edge-length residuals as defined in section 2.3.4.1, and shows a good match with the target even though non-zero values exist in the deformed grid, particularly at high aspect ratio regions mainly due to the limited number of nodes or edges available for stretching and compressing. Final oscillations of the average and maximum force plots are also results of this restraint because toward the end of the iterations, whenever the ball-vertex method fixes one region of the grid, the quality of another region deteriorates, causing unintended fluctuations (Figures 67, 65).

On the other hand, the initial maximum force magnitude experiences a good decline approximately in the range of $80 - 100\%$ as the technique almost immediately transforms the grid into an interim state that is the nearest to the target with the given number of total node points. If that number is inadequate, little can be changed after a few iterations, as exemplified in Figures 65 and 67 for the average and maximum forces, and residual f_K.

$$\mathbf{h_x} = \begin{cases} -0.16x - 0.3 & if x \in [-5.0, -2.5] \\ 0.16x + 0.5 & if x \in [-2.5, 0.0] \\ -0.16x + 0.5 & if x \in [0.0, 3.0] \\ 0.36x - 0.8 & if x \in [3.0, 5.0] \end{cases} \quad \mathbf{h_y} = \begin{cases} -0.16y - 0.3 & if y \in [-5.0, -2.5] \\ 0.16y + 0.5 & if y \in [-2.5, 0.0] \\ -0.16y + 0.5 & if y \in [0.0, 3.0] \\ 0.36y - 0.8 & if y \in [3.0, 5.0] \end{cases} \tag{93}$$

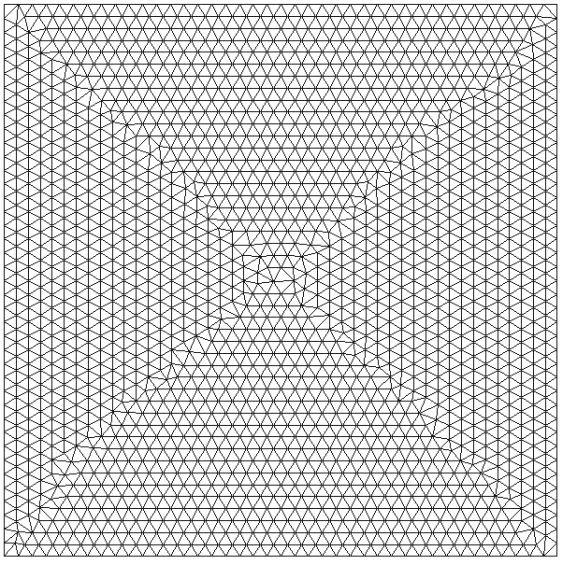
$$\mathbf{h_x} = \begin{cases} -0.18x - 0.4 & if x \in [-5.0, -2.5] \\ 20^{\frac{0.924x - 0.69}{3}} & if x \in [-2.5, 0.0] \\ 5^{\frac{-x-1.29}{3}} & if x \in [0.0, 3.0] \\ 0.1 + \frac{(x-3.1)^4}{20} & if x \in [3.0, 5.0] \end{cases} \quad \mathbf{h_y} = \begin{cases} -0.18y - 0.4 & if y \in [-5.0, -2.5] \\ 20^{\frac{0.924y - 0.69}{3}} & if y \in [-2.5, 0.0] \\ 5^{\frac{-y-1.29}{3}} & if y \in [0.0, 3.0] \\ 0.1 + \frac{(y-3.1)^4}{20} & if y \in [3.0, 5.0] \end{cases} \tag{94}$$

In order to overcome this problem, one must either add new nodes through node insertion and topological changes (h-adaptation) or start with sufficient number of nodes. The next example problem of a circular deformation starts with sufficient number of nodes, resulting in nearly zero residual distribution and smooth convergence of the force field. In this example, the purpose is to deform the original mesh such that we obtain a circular cluster centered at $(0, 0)$ with a radius of three which is successfully done, as shown in Figures 68, 69, and 70. In this non-aligned clustering, element sizes are defined as in equation (95), where x and y are the Cartesian coordinates, h_1 and h_2 are the target sizes along the principle axes, and R is the radius that is equal to three. For this clustering the target metric expression (eqn. 96) has a rotational component R_θ.

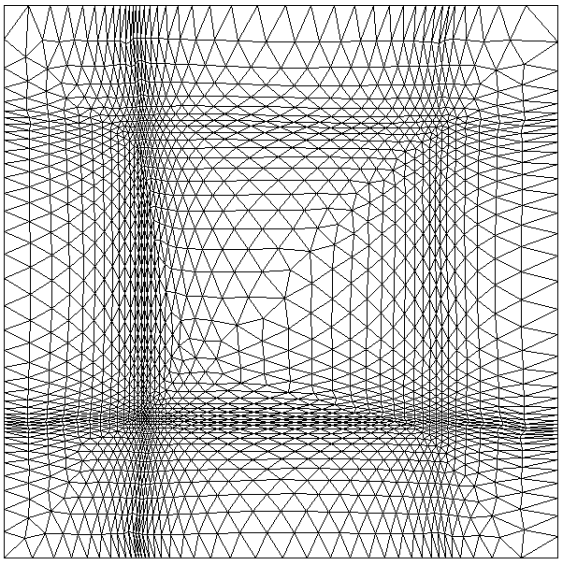
We emphasize that although the initial mesh of the current example is the same as the initial mesh of the previous examples, the difference in the final grids is substantial. The rotational component underlies the uniqueness of the particular example because elements have to be both stretched and rotated, as illustrated in Figure 69.

Once again, the last three examples prove that metrics are simple and effective tools in both adaptive and dynamic meshing. Furthermore, it is noticeable that the converged force field is not absolute zero. In fact, both the maximum and average values converge to a finite non-zero value. The non-zero force field simply means that the absolute target will not be reached even though the average residuals are already stationary.

$$r = \sqrt{x^2 + y^2}, \qquad \theta = \arctan\frac{y}{x}, \qquad h_1 = \frac{1}{240} + \frac{1}{6}|R - r|, \quad h_2 = \frac{r}{40} + \frac{1}{7}|R - r| \tag{95}$$



(a) Initial mesh



(b) Deformed mesh

Figure 63: Linear deformation of a two-dimensional isotropic mesh

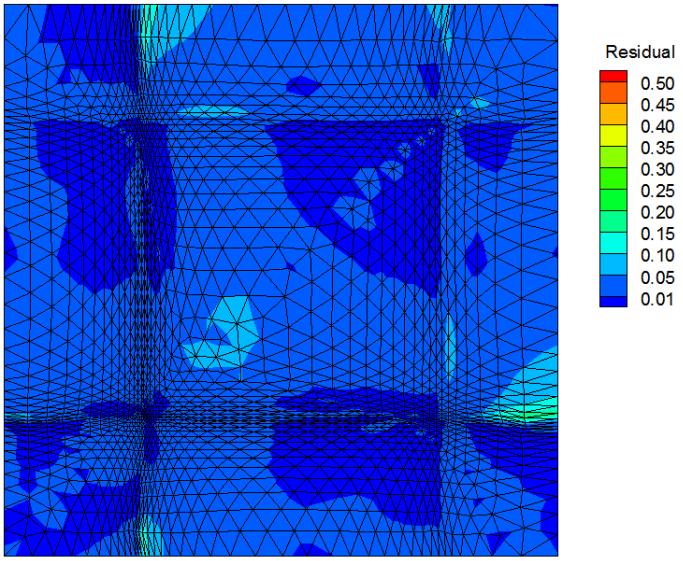
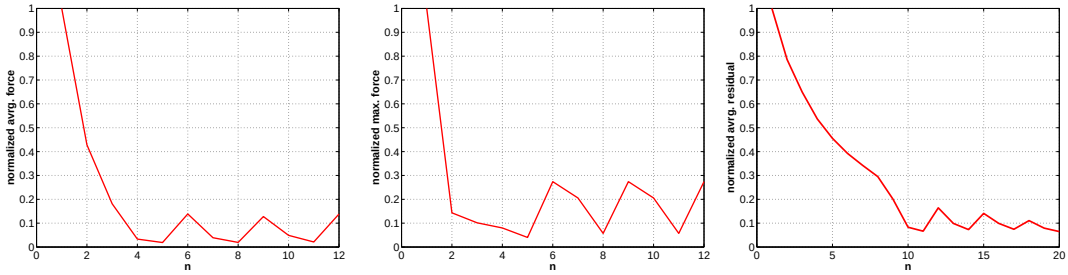


Figure 64: Linear deformation of a two-dimensional isotropic mesh: residual distribution



(a) Normalized average force (b) Normalized maximum force (c) Norm. avrg. residual f_K

Figure 65: Force convergence of the linear deformation of a two-dimensional isotropic mesh

$$\bar{M} = R_\theta \begin{pmatrix} 1/h_1^2 & 0 \\ 0 & 1/h_2^2 \end{pmatrix} R_\theta^{-1}, \qquad R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{96}$$

4.5.2 Vertex Repositioning with A-posteriori-Defined Metrics

In recent years, considerable effort has gone into mesh refinement near shock [56], [70]. However, substantial improvement can also be made in shock resolution by repositioning the near-shock nodes. For this purpose, we solve a simple oblique shock problem on a wall

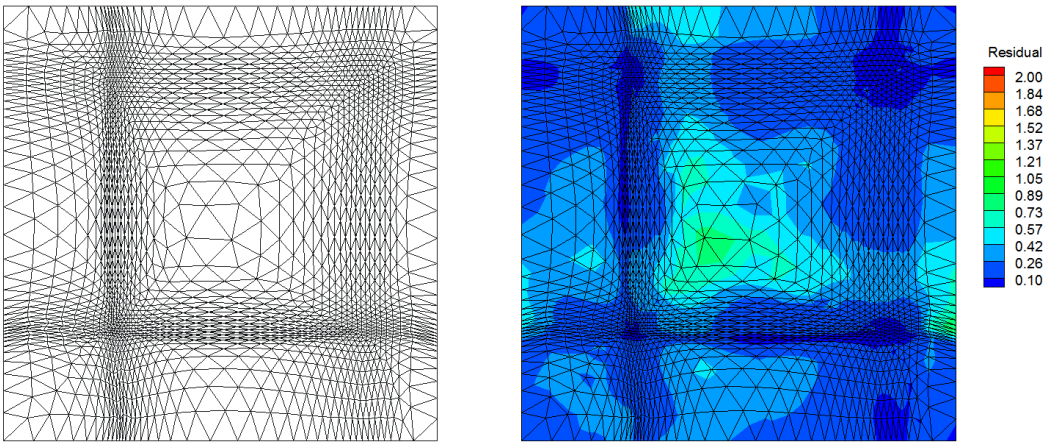
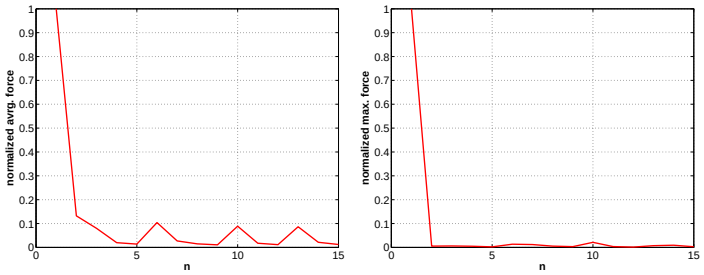


Figure 66: Exponential deformation of a two-dimensional isotropic mesh



(a) Normalized average force (b) Normalized maximum force

Figure 67: Force convergence of the exponential deformation of a two-dimensional isotropic mesh

in which the inflow angle and the Mach number are set at -25^o and 2.25, respectively. The solution is obtained by using a two-dimensional Navier-Stokes solver, NSC2KE, an open software developed by INRIA [56]. NSC2KE employs a finite volume Galerkin method on unstructured grid flows. In order to solve the Euler part of the equations, it has Roe, Osher and Kinetic solvers available while for turbulent flows it has a built-in $k - \epsilon$ model. In the current example, we solve the shock problem with Euler equations where fluxes are calculated using the first order Roe upwind scheme, and explicit time integration is obtained through fourth order Runga-Kutta method. Although this example seems elementary, it is a non-trivial test case that involves the right conversion of the flow gradients into the accurate target metrics, which is crucial in a sense that the success of the adaptation scheme

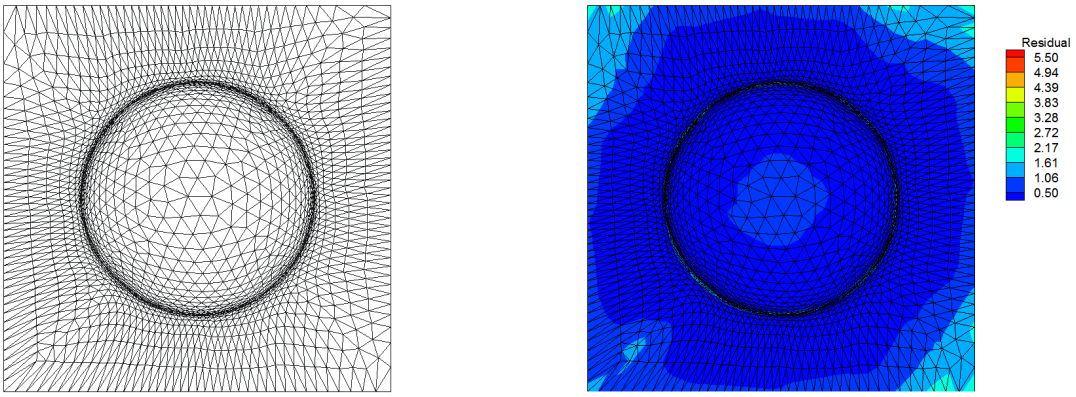
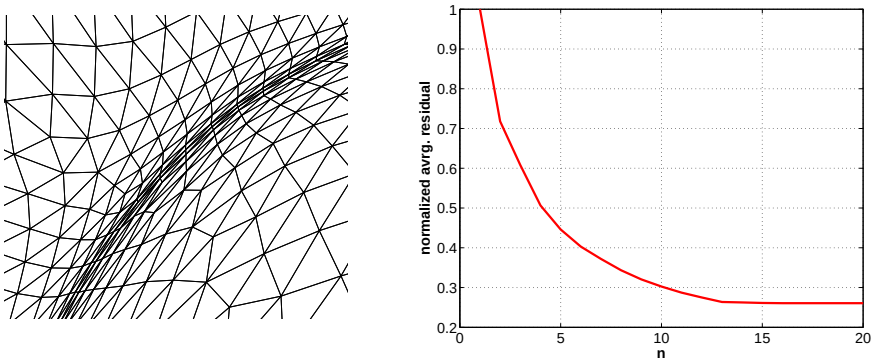


Figure 68: Circular deformation of a two-dimensional isotropic mesh: $R = 3$

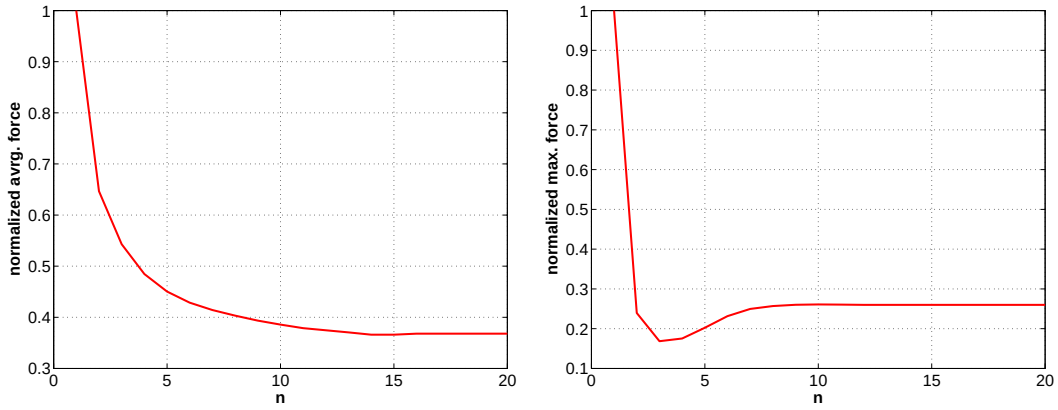


(a) closer look (b) residual (f_K) history

Figure 69: Circular deformation of a two-dimensional isotropic mesh: a closer look and average residual history.

is determined based on this ability.

Following the procedure outlined in Figure 71, we start by putting the initial mesh, created by software EMC2 [70], into the flow solver to obtain a first solution. The homogeneous initial mesh is clearly not capable of resolving the shock as shown in Figure 72, in which shock thickness is unacceptably large. Next, we extract the second derivatives, or the Hessian information, from the Mach solution as detailed in section 2.3.3. We feed the Hessian into the metric-based mesh deformation routine, manipulating the Hessian to get the target metric distribution. It is worth mentioning here that at each internal deformation step, the initial Hessian distribution must be projected from a background grid onto



(a) Normalized average force (b) Normalized maximum force

Figure 70: Force convergence of circular deformation of a two-dimensional isotropic mesh

the current grid locations through a search algorithm. The current mesh is then deformed accordingly, and one cycle of adaptation is completed by solving the shock problem with the newly adapted mesh. In each cycle, the flow solver NSC2KE is let run until steady state solution is obtained, which happens when the flow residual decreases by four orders of magnitude. This process is repeated until either a satisfactory flow solution is obtained or a convergence is reached. During intermediate iterations, it is usually sufficient to run the mesh deformer about five to ten steps before putting the mesh into the flow solver again. Figures 72 through 75 exhibit the evolution of the solution with the adapted meshes. It is remarkable that nodal density increases at locations where Mach counters are concentrated whereas the nodal density decreases in low-gradient regions, which confirms the accuracy of the adaptation process. After only three cycles, the shock resolution is considerably superior to that of the initial, shown in Figure 75. As initial mesh indicates, the user does not have to make any presumptions about the possible solution, the shock location, or the required mesh field.

Furthermore, it is noticeable that as the nodes get clustered near the shock, other nodes away from the shock stretch out, forming larger sized triangles. This is mainly because the gradients of the flow vanish in those regions, resulting in nearly zero second derivatives and therefore infinitely large edge lengths (eqn. 13). However, it is physically not possible to have non-finite edge lengths in a finite domain. Thus, for regions of uniform solution,

it is general practice to set the target metric to $\bar{M} = cI$, where I stands for the identity matrix, and c is a domain-dependent constant. In the rest of the domain, the target metric is described as in equation (15), $\bar{M} = [I + Ch_{norm}R|\Lambda|R^{-1}]$, where we choose $c = 25$ and $C = 500$ for this problem. The Hessian has to be modified as in the above equation in order to get reasonable target element sizes because the Hessian can provide only the ratio of the maximum to minimum edge lengths [60].

Figure 76 illustrates the final residual distribution. Even though in the vicinity of the shock, high metric-edge-length residuals still exist in the final adapted mesh, the flow solution is satisfactory enough to halt the iterations. Once again, topological modification would further reduce residual magnitudes; but it may not be necessary, as shown in this particular example.

As explained early in the discussion, fine tuning of the constants c and C in the modified equations is important. Through Figures 77 to 80, we investigate the sensitivity of the adapted meshes to these constants. The constant c, ranging from 0.5 to 25, appears to control the element sizes in the regions of uniform flow (Figure 77), where smaller c values result in larger elements that eventually push more nodal points into the shock clustering. On the other hand, constant C ranging from 1 to 10,000 appears to have a stronger control on element sizes and thickness of clustering near the shock (Figure 79), where non-zero flow gradients are present. Variations in constant C, also affect the edge lengths as well as the nodal density behind the shock. This influence is more obvious especially at higher C values. Furthermore, figures of mach contours reveal that except for extremely large C values (i.e., much larger than 500), most C values produce reasonably well mach resolutions.

As the values of the constants get closer to $C = 500$ and $c = 5$, both the shock resolution and the adapted mesh improves considerably as shown in Figures 78 and 80. Therefore, for this simple shock problem, parameters $C = 500$ and $c = 5$ seem to be the optimum values, while for other type of problems optimum parameters would be different.

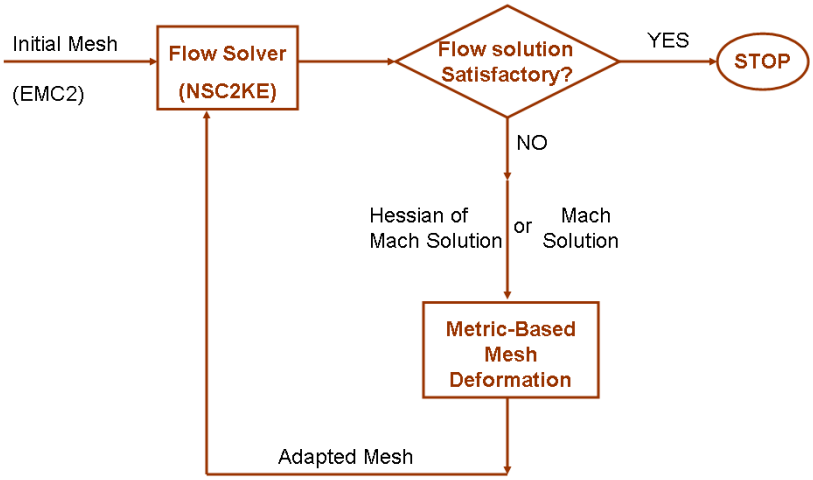


Figure 71: Hessian-driven mesh deformation: flowchart.

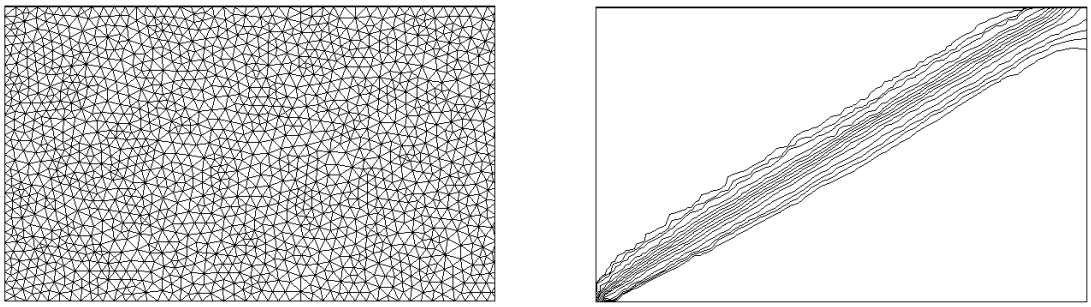


Figure 72: Initial, non-deformed mesh and initial Mach solution of the simple shock problem

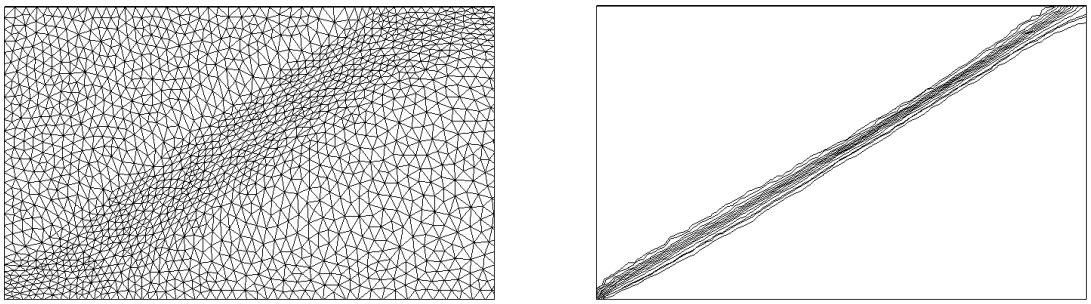


Figure 73: Deformed mesh and Mach solution of the simple shock problem: after cycle 1.

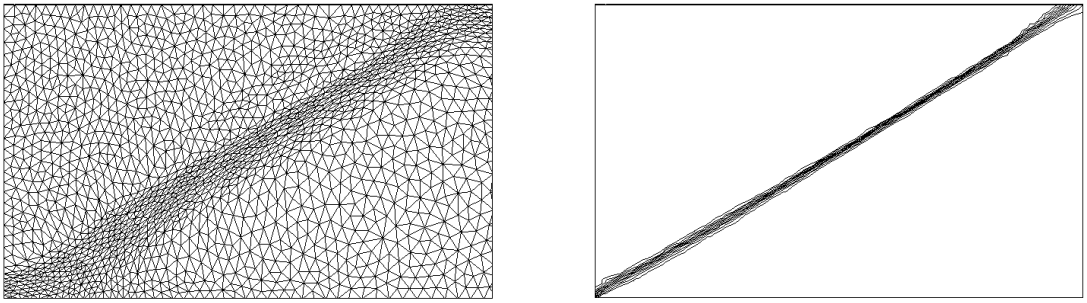


Figure 74: Deformed mesh and Mach solution of the simple shock problem: after cycle 2.

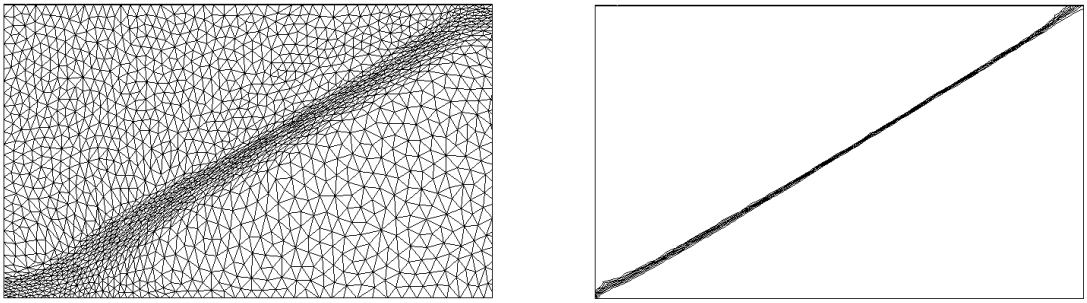


Figure 75: Deformed mesh and Mach solution of the simple shock problem: after cycle 3.

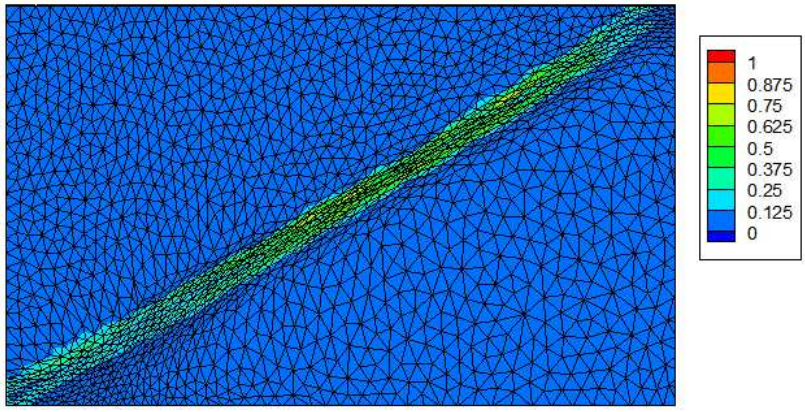
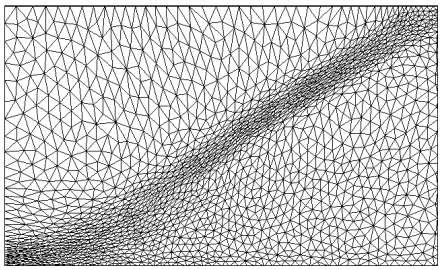
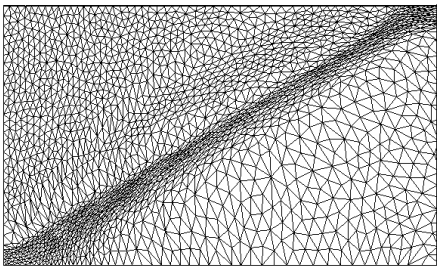


Figure 76: Residual f_K distribution of the shock problem after r-adaptation.

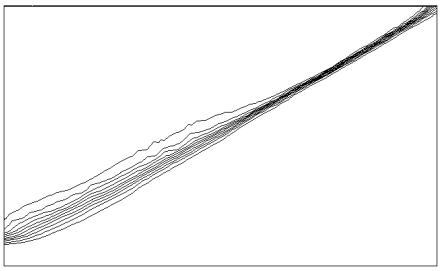


(a) c=0.5, C=500

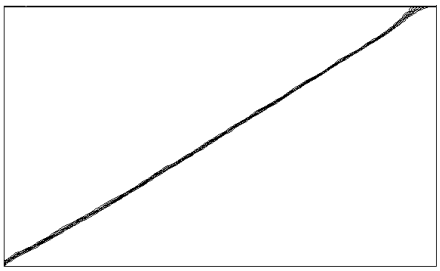


(b) c=25, C=500

Figure 77: Adapted meshes with various c values.

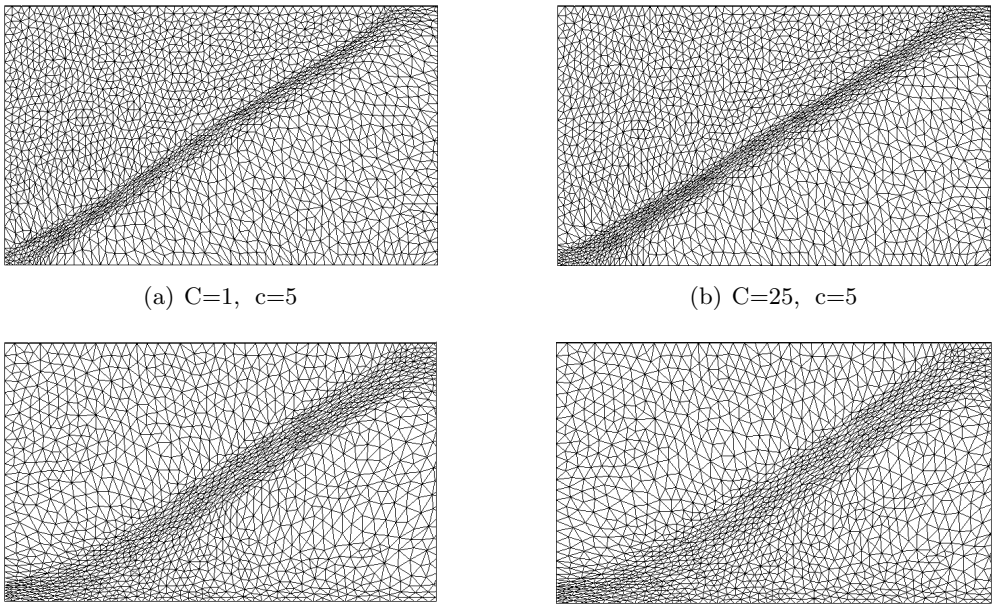


(a) c=0.5, C=500



(b) c=25, C=500

Figure 78: Mach solutions of adapted meshes with various c values.



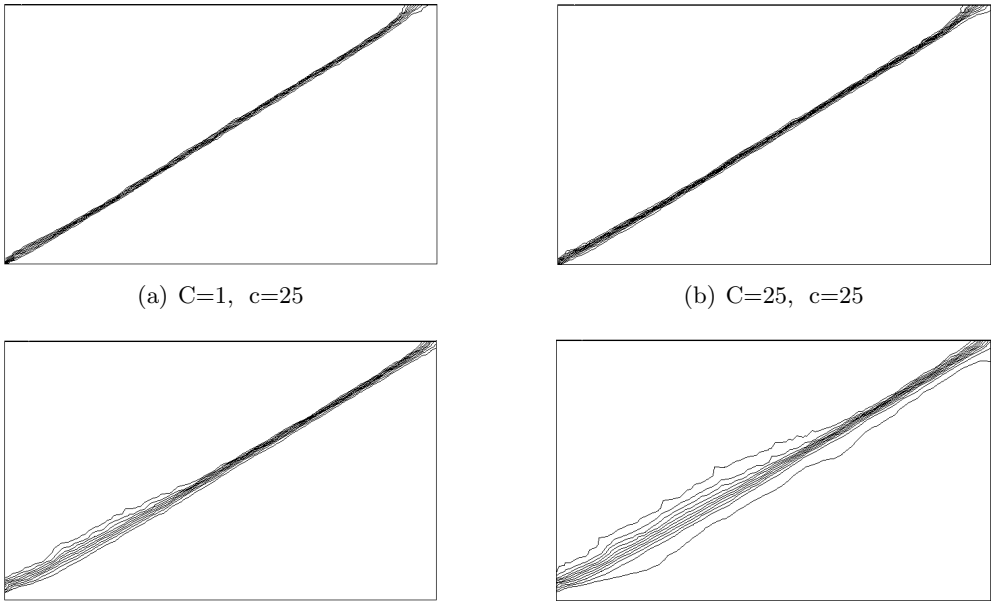
(a) C=1, c=5

(b) C=25, c=5

(c) C=5,000, c=5

(d) C=10,000, c=5

Figure 79: Adapted meshes with various C values.



(a) C=1, c=25

(b) C=25, c=25

(c) C=5,000, c=25

(d) C=10,000, c=25

Figure 80: Mach solutions of adapted meshes with various C values.

Chapter V

CONCLUSIONS AND FUTURE WORK

5.1 Summary and Conclusions

In the first half of this work, we considered the problem of optimizing a grid so that it conformed to a given metric. The analysis started with a definition of the compliance of a current metric to a target metric, which led to the formulation of a matrix equation whose residual is used for driving the optimization objective function. We preferred to use a combined residual function simply based on the metric edge lengths and the metric inscribed radius of each simplex. Not only does this measure work well in practice, but it is also reasonably cheap to compute.

We then described a Gauss-Seidel optimization algorithm based on the removal of bad elements from the grid. The removal of each element was attempted with several local retriangulation operators that add/remove vertices, retriangulate sets of vertices without changing their location, or move vertices without altering their connectivities.

However, sometimes bad elements still remained at the end of the optimization process, as they were converged to the target metric through a sequence of discrete local mesh modifications with a limited set of tools. Due to such limitations, a lower compliance residual below a given tolerance everywhere over the mesh cannot be guaranteed. Moreover, a strictly decreasing acceptance criterion for the compliance residual makes it even harder to reach a residual function below the specified tolerance.

On the basis of these considerations, we introduced a statistical approach that relaxes the initial acceptance criterion, allowing a number of new mesh configurations that otherwise would have been rejected. This approach, called "simulated annealing", has been widely used in studies for improving the efficiency of many optimization procedures. It also helps to find global minima that otherwise would be unreachable due to the non-convex nature of many problems. For example, for the anisotropic problems we investigated, the

improvement in the worst element quality was as high as 67-times, while in a similar effort, triangulation of an high aspect-ratio wing improved by 5-fold compared to the classical approach which had previously resulted in unacceptably poor mesh elements at the leading edge. Although simulated annealing approach could be quite time consuming (up to 3-fold more than the classical approach, Section 2.5.2), our studies have demonstrated that it represents a powerful tool if locally applied, i.e., used in some portions of the domain where elements are much worse compared to other poorly-shaped elements. This local approach decreased the computational cost to the same level as the greedy algorithm, keeping the remarkable mesh quality of the global simulated annealing scheme (Section 2.5.3).

Moreover, a statistical study based on simulated annealing parameters was conducted in order to investigate the effects of parameter selection. For most optimization problems using SA, fine tuning is a key to finding a solution. We experimented with the initial temperature, the temperature schedule, and acceptable moves per temperature. Observations revealed that both SA and LSA are robust in terms of their parameters unless they are chosen at extremities, e.g., very low initial temperatures such as $\theta_{init} = 0.0001$. In addition, we examined sensitivity to initial conditions, i.e., the initial mesh. Although the greedy algorithm results in a variety of optimized meshes with different initial conditions, the SA and LSA algorithms are able to transform any initial mesh into the same superior quality final mesh. For example, an initial mesh with as few as 6,528 tetrahedra was transformed to an high quality mesh with 0.1 millions elements for the wing problem.

In the second half of this study, we presented a simple method for enhancing the robustness of pseudo-structural techniques for the deformation of unstructured quadrilateral/hexahedral and triangular/tetrahedral meshes. In fact, the method is applicable to meshes in all categories such as prism, pent, and pyramid because its node-oriented formulation is based on an edge database. In this sense, the method that we called as "ball-vertex" is the first method in dynamic mesh-deformation literature that can cover all different types of grid elements for large amplitude boundary movements. Similar to the torsional spring method [34, 31], the pseudo-structural system is designed in such a way as to avoid the appearance of collapse mechanisms for the mesh elements.

The method works as follows: each vertex is confined to its ball with additional linear springs added in a straightforward manner to the network of edge springs. These springs connect each vertex to virtual points on the diagonal obtained in non-simplicial topologies by projecting the vertex onto the diagonal of the quadrilateral or onto the diagonal face of hexahedra.

We compared the new method with the spring analogy technique [14]. Numerical examples in two dimensions showed that the new method behaves well and is able to guarantee nicely graded, good quality meshes even after repeated cycles of very large amplitude deformations such as 30^o pitching and 0.22-chords plunging amplitudes for a NACA0012 airfoil. However, for the same cases, the spring analogy failed almost immediately by creating invalid elements, proving that the ball-vertex method is a superior device that offers a simple solution to severe mesh deformation problems. It is effective not only for open boundary problems, e.g., pitching airfoil in ambient air, but also for problems with relative motion of closely located solid bodies, e.g., compression-extension of a flap or a slat. The latter is a particularly difficult problem because the potential energy stored in the springs can not be easily dissipated into far field boundaries. For example, a multi-element airfoil with a flap deflection angle 34^o and extraction distance of 0.15 chords, and a slat extraction of 0.1032 chords, successfully underwent tremendous amount of deformation with the ball-vertex method while keeping a well-behaved periodic boundary motion (Section 3.8.4). This type of extreme deformations near the solid boundaries were not achieved in literature before without remeshing.

Later, the ball-vertex technique was extended into three dimensions and tested for a pitching and plunging wing, as well as a bending wing at extreme deformations. Once again, the ball-vertex technique proved robust and reliable in three dimensions. Three-dimensional applications involved hexahedral grids because most of the CFD and CSD (computational solid dynamics) numerical simulations made use of structural grids, which are hexahedral in nature, because of their superiority over tetrahedral grids in accuracy. The technique proved very effective even for non-rigid deformations such as bending of a wing, where at the peak of its motion the tip was vertically displaced by half of half-span, a very large

displacement considering the local grid sizings. So with this example, it is shown that the ball-vertex method could be a quite useful tool in flutter analysis that involves bending type of mesh deformations where structures are subjected to extreme boundary motions. Furthermore fluid-structure-interaction simulations are also in great need of such robust mesh deformation tools.

In the final part of the study, we introduced a novel mesh-motion technique based on the ball-vertex method, adapting an initial mesh into a target mesh whose characteristics are prescribed by a given metric. The technique works as follows: first, it computes a force field over the domain that will, upon application, transform a current mesh into a target mesh. Essential displacements and therefore the corresponding force vectors are extracted from the information about current and target metrics. Then the net of edge and face springs are deformed with the ball-vertex method by applying forces at the vertices. This process-cycle is repeated until the average and/or maximum force is equal to zero or to a certain user-defined limit, or until a satisfactory numerical solution is reached.

This study also discussed several examples of a-priori metric definitions, both aligned and non-aligned along axes. The force field converged just in several iterations. Specifically, maximum force decreased by $80-90\%$ at the first iteration allowing a fast adaptation of the mesh just in few iterations. The resultant meshes complied very well with the target, based on compliance residual assessments. We obtained good agreement, even for the non-aligned elements of the meshes.

As for the a-posteriori metric definition, we used the Hessian recovery technique that obtains discrete metric distribution from second derivatives of the flow solution using NSC2KE, a flow solver. A course isotropic mesh that is incapable of accurately resolving an oblique shock was successfully adapted using the metric-driven mesh deformation technique. After only three iterations between the mesh deformer and the flow solver, a very accurate shock resolution was effectively obtained. So, the scheme dramatically reduced the user intervention by completely automating the flow solution, and removing errors that can be caused by inaccurate initial judgements of the user that result in poor resolution of the flow properties.

5.2 Recommended Future Work

Metric-based mesh deformation with the ball-vertex method has proven to be an effective and robust solution to adaptation problems. The method basically answers an optimization problem: given the topology and total number of vertices, the technique progresses toward finding the optimal distribution of the nodes that will comply best with the target mesh or result in a more accurate flow solution.

Clearly, H-adaptation can benefit tremendously from this type of powerful mesh repositioning tool. Incorporating metric driven mesh motion as another locally applied primitive would constitute a coherent and even more powerful mesh adaptation (hr-) and generation method. One could rightfully expect that the error levels to become lower than each individual adaptation technique is capable of decreasing, because both relative positioning of nodes and their connectivity play major roles in numerical simulations.

Furthermore, implementing the boundary-driven mesh motion procedure within this hr-adaptation technique would enable local optimization to be used whenever needed. For example, when the boundary motion fails creating invalid elements, with local modifications one can remesh and continue the simulation, which would have to be stopped otherwise. Therefore the three procedures outlined in this thesis can be harmonized and consolidated in a single tool, which would have much wider applicability.

Considering the sizes and complexities of the meshes used nowadays and will be used in the future, the proposed hr-adaptation method also provides an economical way to adapt meshes. For example, when only h-adaptation is applied all along, some redundant point insertions and removals are expected to happen.

Lastly, from the theoretical point of view it is recommended to a) formally prove that all collapse mechanisms are avoided by the ball containment condition, b) extend afterwards the condition to continuous mesh motion methods by linking ball-vertex springs lengths to some material property and avoid by design the appearance of collapse mechanisms even with such formulations.

Appendix A

MESH VALIDITY AND GEOMETRIC MODEL INTERFACE

An automatic mesh generator must interact with the geometric model of the domain being meshed. Reliability of the mesh generator depends on these interactions. The procedures described in Chapter 2 are interfaced with a computer-aided design (CAD) system called Parasolid. The interface [76, 75] provides geometric and/or topological interrogations through a limited set of functional operators, that hide the details of the data structure and implementations used by the CAD system from the present application. The integration of the meshing procedure with a geometric modelling system demands the creation of only certain geometric queries and requires no knowledge of the mesh generator or the data structures internal to the geometric modelling system. The only requirement is a knowledge of the routines available from the model that is used to build specific geometric queries.

All mesh modification operations mentioned in Section 2.3.6 should be topologically as well as geometrically validated using the interface. The resulting procedures guarantee the validity of the generated grid in domains of arbitrary complexity. Before explaining how these validation checks are performed, we will describe in detail the geometric model, mesh representations, and classifications.

A.1 Geometric Model

Since mesh generation is concerned with the decomposition of a geometric domain into a union of simple geometric entities, the definition of a valid mesh must include the definition of the geometric domain. A geometric model [76] can be considered to be an object representation in terms of its geometry G, topology T, and associated attributes A.

$$G = ({}_{G}T,\ {}_{G}G,\ {}_{G}A) \tag{97}$$

where ${}_{G}G$ represents the geometric information defining the shape of the entities that describe the domain, and ${}_{G}T$ represents the topological types and adjacency relations among

the various entities of the object.

A.1.1 Topology

The topological entities ${}_GT$ are represented by the $0-D$ vertices T^0, the $1-D$ edges T^1, the $2-D$ faces T^2, and the $3-D$ regions T^3. These entities relate to one another in an orderly and hierarchical manner (Figure 81) [76, 75]. For example, located at the boundary of a region T^3 is an ordered set of faces; at the boundary of a face T^2, an ordered set of edges; at the boundary of an edge T^1, an ordered set of vertices. When employing a functional interface, such order is needed for a general representation of the geometric model. For example, one might need to know which regions are on each side of the face or similarly which edges are pointing out from a certain node. Representing such a situation requires both upward and downward adjacencies for each entity, as depicted in Figure 81.

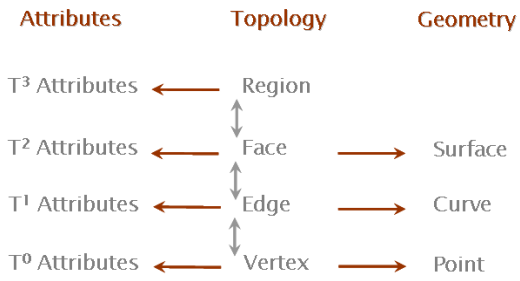


Figure 81: Hierarchical representation of the geometric model

A.1.2 Geometry

The geometry ${}_GG$ has information about the shape and space location of a topological entity [76]. It is an attribute of topology. For example: the geometry associated with a vertex T^0 is the space coordinates of a point P; the geometry associated with an edge T^1 is the mathematical representation of a curve (a B-spline, etc.); the geometry associated with a face T^2 is the mathematical representation of a surface.

A.1.3 Attributes

All the additional descriptive information associated with the problem are stored in attributes ${}_GA$, such as color, boundary conditions, material properties, and mesh density

[76].

A.2 Meshes

Meshes can be represented in a way similar to geometric models [76, 75].

$$M = ({}_M T,\ {}_M G,\ {}_M A) \tag{98}$$

The topological entities are vertices ${}_M T^0$, edges ${}_M T^1$, and faces ${}_M T^2$, the regions ${}_M T^3$ of the mesh. The geometric entities are points P associated with the vertices, curves associated with the edges, and surfaces associated with the faces. The attributes are represented by all the other information that defines the particular analysis. Again, similar to geometric model, a hierarchical order exists among the different entities of the mesh.

A.2.1 Classification

A classification is defined as the unique association of a topological mesh entity of dimension d_i, ${}_M T_i^{d_i}$, to a topological model entity of dimension d_j, ${}_G T_j^{d_j}$ and denoted [76] by

$${}_M T_i^{d_i} \subset {}_G T_j^{d_j}, \quad d_i \leqslant d_j \tag{99}$$

where the classification symbol, $\subset$, shows that the entity or set on the left is classified on the entity on the right. For example, vertices ${}_M T^0$ can be classified on ${}_G T^3$, ${}_G T^2$, ${}_G T^1$, ${}_G T^0$ (Figure 82), and edges ${}_M T^1$ on ${}_G T^3$, ${}_G T^2$, ${}_G T^1$ (Figure 83). Similarly, faces ${}_M T^2$ can be classified on ${}_G T^3$, ${}_G T^2$ (Figure 84) while regions ${}_M T^3$ can be classified only on ${}_G T^3$. It is also possible to classify multiple mesh entities on a single model entity.

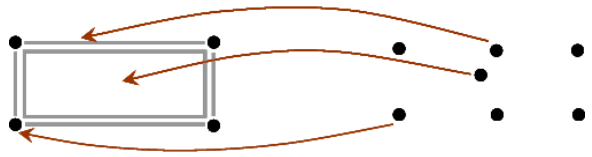


Figure 82: Classification of vertices on the model. Left: model ${}_G T^j$; right: mesh ${}_M T^i$

Figure 83: Classification of edges on the model. Left: model ${}_GT^j$; right: mesh ${}_MT^i$

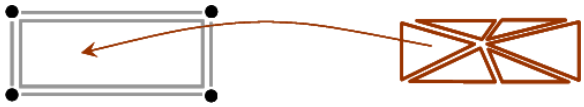


Figure 84: Classification of faces on the model. Left: model ${}_GT^j$; right: mesh ${}_MT^i$

A.3 Mesh Validity

Given a set P of M points in n dimensions, each classified with respect to a geometric model G, an n-dimensional geometric triangulation T_G^n is the set of N n-dimensional non-degenerate simplexes s^{d_i}

$$T_G^n = s_1^{d_1}, s_2^{d_2}, , s_N^{d_N}$$

with $0 \leq d_i \leq n$, a valid mesh is defined [76, 75] such that

1. All the vertices of each $s_i^{d_i} \subset P$
2. $interior(s_i^{d_i}) \cap interior(s_j^{d_j}) = 0 \; \forall \; i \neq j$
3. T_G^n is topologically compatible with G
4. T_G^n is geometrically similar to G

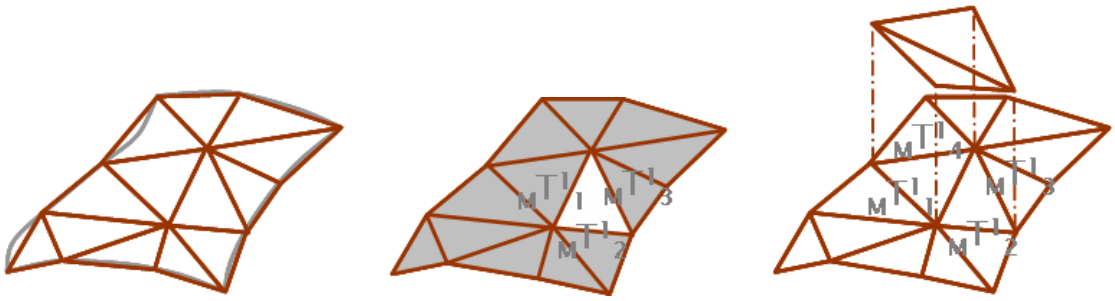
A.3.1 Topological Compatibility

Definition: Consider a mesh with all its ${}_MT^0$ vertices classified according to the geometric model. Let ${}_MT^d$ be the remaining entities of order $1 \leq d \leq n$ with boundary entities $\partial({}_MT^d)$. Consider an entity of the geometric model ${}_GT^{d_j}$ with boundary entities $\partial({}_GT^{d_j})$. The mesh is topologically compatible with ${}_GT^{dj}$ [76, 75] if

1. Each $\partial({}_MT_i^d) \subset \; {}_GT_j^d$ is used by 2 ${}_MT_k^d \subset \; {}_GT_j^d$
2. Each $\partial({}_MT_i^d) \subset \partial({}_GT_j^d)$ is used by 1 ${}_MT_k^d \subset \; {}_GT_j^d$

The first condition requires the following: Each edge, which is bounding a face and classified on the model, must be used by exactly two faces; while each face, which is bounding a region and classified on the model, must be used by two regions of the mesh. A similar relation can be derived easily between vertices and the edges they are bounding. The second condition, however, requests that each bounding entity classified on the model boundary must be used by exactly one entity of the mesh that is categorized one level up hierarchically. For example, an edge classified on the model boundary can be used by only one face of the mesh. Again, a face of the mesh classified on the boundary can be used by only one region of the mesh. When satisfying the compatibility of the entire mesh, all of its topological entities are checked for their compatibility with the geometric model.

Consider the valid triangulation in Figure 85(a). Each edge ${}_MT^1 \subset {}_GT^2$ is used by 2 faces ${}_MT^2$ (Condition 1), and each edge ${}_MT^1 \subset$ on an edge $\partial({}_GT^2)$ is used by 1 face ${}_MT^2$ (Condition 2). Next, in Figure 85(b), a topological hole is characterized by the fact that



(a) Topologically compatible faces (b) Topological hole (c) Topological redundancy

Figure 85: Topological compatibility

three edges ${}_MT_1^1$, ${}_MT_2^1$, ${}_MT_3^1$ are $\subset$ ${}_GT^2$, but are used by one single face ${}_MT^2$ only, which violates the first condition.

Similarly, if mesh entities are used too often, topological redundancy is generated, as in Figure 85(c), in which the edges ${}_MT_1^1$, ${}_MT_2^1$, ${}_MT_3^1$, ${}_MT_4^1$ are $\subset$ ${}_GT^2$, but are used by three faces. Again, condition 1 is violated.

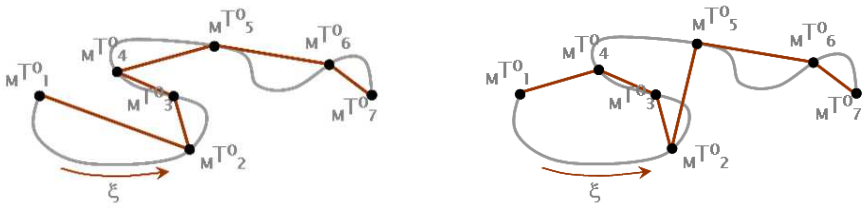
It is possible to remove topological holes by creating a correct connection or inserting new nodal points in the area of the hole. In the same manner, redundancies can be treated by an appropriate reclassification [76].

A.3.2 Geometric Similarity

A geometrically similar mesh exactly matches the geometric domain in the limit of refinement [76]. A set of mesh entities ${}_MT^d = \bigcup_{i=1}^{N} {}_MT_i^d$ is geometrically similar to an entity of the geometric model ${}_GT_k^d$ when

1. Each ${}_MT_i^d \subset {}_GT_k^d$
2. The parametric intersection ${}_MT_i^d \cap_M^* T_j^d = 0 \;\; \forall\, i \neq j$

The concept of geometrical similarity is demonstrated by the curve mesh example in Figure 86, in which parametrization is defined on ${}_GT^1$ with $0 \leq \xi \leq 1$.



(a) Satisfying geometric similarity (b) Violating geometric similarity

Figure 86: Geometric similarity on a model edge.

The mesh entities in Figure 86(b) do not satisfy the geometric similarity conditions because four mesh edges ${}_MT_1^1$, ${}_MT_2^1$, ${}_MT_3^1$ and ${}_MT_4^1$ overlap in the parametric space ξ of the model edge. On the other hand, mesh edges in Figure 86(a) do satisfy both similarity requirements.

By employing an edge parameter ξ, the mesh vertices located on the model edge can be sorted in order [76]. The mesh is compatible and geometrically similar if a single edge connects each pair of vertices, and these edges do not intersect each other. Mesh edges that connect non-consecutive vertices are redundant and require correction.

Likewise, while geometric similarity of the faces can be checked through a local surface parametrization, region similarity and compatibility are obtained by means of inheritance [76]. That is, once all the faces are compatible with the model, regions bounded by these faces are automatically validated.

REFERENCES

[1] Aarts, E. and Lenstra, J., *Local Search in Combinatorial Optimization*. Chichester: J. Wiley &Sons, 1997.

[2] Acikgoz, N. and Bottasso, C., "A local simulated annealing strategy for mesh optimization," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, NV), January 2006.

[3] Acikgoz, N. and Bottasso, C., "A new mesh deformation technique for simplicial and non-simplicial meshes," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, NV), January 2006.

[4] Acikgoz, N. and Bottasso, C., "Metric-driven mesh optimization using a local simulated annealing algorithm," *International Journal for Numerical Methods in Engineering*, doi:10.1002/nme.1904.

[5] Acikgoz, N. and Bottasso, C., "A unified approach to the deformation of simplicial and non-simplicial in two and three dimensions with guaranteed validity," *Journal of Computers and Structures*, doi:10:1016/j.computstruc.2006.11.009.

[6] Acikgoz, N., Bottasso, C., and Detomi, D., "Metric based mesh optimization using simulated annealing," tech. rep., European Congress on Computational Methods in Applied Sciences and Engineering, ECOMMAS, Finland, 2004.

[7] Amenta, N., Bern, M., and Eppstein, D., "Optimal point placement of mesh smoothing," *Journal of Algorithms*, vol. 30, pp. 302–322, 1998.

[8] Axelsson, O., *Iterative Solution Methods*. Cambridge: Cambridge University Press, 1996.

[9] Babuska, I. and Rheinboldt, W., "A-posteriori error estimates for the finite element method," *International Journal for Numerical Methods in Engineering*, vol. 12, pp. 1597–1615, 1978.

[10] Baines, M., Leary, S., and Hubbard, M., "Multidimensional least squares fluctuations distribution schemes with adaptive mesh movement for steady hyperbolic equations," *Journal on Scientific Computing*, vol. 23, no. 5, pp. 1485–1502, 2002.

[11] Bank, R. and Smith, R., "Mesh smoothing using a-posteriori error estimates," *SIAM J. Numer. Anal.*, vol. 34, pp. 979–997, 1996.

[12] Bar-Yoseph, P., Mereu, S., Chippada, S., and Kalro, V., "Automatic monitoring of element shape quality in 2-d and 3-d computational mesh dynamics," *Computer Mechanics*, vol. 27, pp. 378–395, 2001.

[13] Barrett, R., Berry, M., and Chan, T., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphia: SIAM, 1994.

[14] Batina, J., "Unsteady euler airfoil solutions using unstructured dynamic meshes," Tech. Rep. AIAA 89-0150, 27th Aerospace Sciences Meeting, Reno, NV, 1989.

[15] Bauchau, O., *Class notes of AE6263, Flexible Multibody Dynamics*. http://www.ae.gatech.edu/people/obauchau/flexmb/FlexMb.pdf: School of Aerospace Engineering, Georgia Institute of Technology, 2005.

[16] Becker, R., Hansbo, P., and Larson, M., "Energy norm a posteriori error estimation for discontinuous galerkin methods," *Computer Methods in Applied Mechanics and Engineering*, vol. 192, pp. 723–733, 2003.

[17] Bochev, P., Liao, G., and Pena, G., "Analysis and computation of adaptive moving grids by deformation," *Numerical Methods fopr PDE's*, vol. 12, pp. 489–506, 1996.

[18] Boroomand, B. and Zienkiewicz, O., "Recovery procedures in error estimation and adaptivity. Part II: Adaptivity in nonlinear problems of elasto-plasticity behaviour," *Computer Methods in Applied Mechanics and Engineering*, vol. 176, pp. 127–146, July 1999.

[19] Borouchaki, H., George, P., Hecht, F., Laug, P., and Saltel, E., "Delaunay mesh generation governed by metric specifications. Part I. algorithms," *Finite Elements Analysis and Design*, vol. 25, pp. 61–83, 1997.

[20] Borouchaki, H., George, P., Hecht, F., Laug, P., and Saltel, E., "Delaunay mesh generation governed by metric specifications. Part II. applications," *Finite Elements Analysis and Design*, vol. 25, pp. 85–109, 1997.

[21] Bottasso, C. L., "Anisotropic mesh adaption by metric-driven optimization," *International Journal for Numerical Methods in Engineering*, vol. 60, pp. 597–639, 2004.

[22] Bottasso, C. L., Detomi, D., and Serra, R., "The ball-vertex method: A new simple spring analogy method for unstructured dynamic meshes," *Computer Methods in Applied Mechanics and Engineering*, (accepted).

[23] Buscaglia, C. and Dari, E., "Anisotropic mesh optimization and its application in adaptivity," *International Journal for Numerical Methods in Engineering*, vol. 40, pp. 4119–4136, 1997.

[24] Cao, W., Huang, W., and Russell, R., "An error indicator monitor function for an r-adaptive finite element-method," *Journal of Computational Physics*, vol. 170, pp. 871–892, 2001.

[25] Carlson, N. and Miller, K., "Design and application of a gradient-weighted moving finite element code II: In two dimensions," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 766–798, 1998.

[26] Carrannanto, P., Storms, B., Ross, J., and Cummings, R., "Navier-stokes analysis of lift enhancing tabs on multi-element airfoils," *Aircraft Design*, vol. 1, pp. 145–158, 1998.

[27] Castro-Diaz, M., Hecht, F., Mohammadi, B., and Pironneau, O., "Anisotropic unstructured mesh adaption for flow simulations," *International Journal for Numerical Methods in Engineering*, vol. 25, pp. 475–491, 1997.

[28] Cifuentes, A. and Kalbag, A., "A performance study of tetrahedral and hexahedral elements in 3D element structural analysis," *Finite Elements in Analysis and Design*, vol. 12, pp. 313–318, 1992.

[29] D.A.Field, "Laplacian smoothing and delaunay triangulations," *Communications and Applied Numerical Methods*, vol. 4, pp. 709–712, 1988.

[30] de Cock, K., "2D maximum lift prediction of a three element airfoil," Tech. Rep. NLR TP-98235, National Aerospace Laboratory, NLR, Netherlands, 1998.

[31] Degand, C. and Farhat, C., "A three-dimensional torsional spring analogy method for unstructured dynamic meshes," *Computers and Structures*, vol. 80, pp. 305–316, 2002.

[32] Dompierre, J., Vallet, M., Bourgault, Y., Fortin, M., and Habashi, W., "Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part III: unstructured meshes," *International Journal for Numerical Methods in Fluids*, vol. 39, pp. 675–702, 2002.

[33] Eduardo, S. and Law, K., "A distributed application of an adaptive finite element method for fluid problems," *Computers and Structures*, vol. 74, pp. 97–119, 2000.

[34] Farhat, C., Degand, C., Koobus, B., and Lesoinne, M., "Torsional springs for two-dimensional dynamic unstructured fluid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 163, pp. 231–2450, 1998.

[35] Felcman, J., "Grid refinement/alignment in 3D flow computations," *Mathematics and Computers in Simulation*, vol. 61, pp. 317–331, 2003.

[36] Frey, P. and George, P., *Mesh Generation: Application to Finite Elements*. Oxford and Paris: Hermes Science Publishing, 2000.

[37] Greenman, R. M. and Roth, K., "Minimizing computational data requirements for multi-element airfoils using neural networks," *Journal of Aircraft*, vol. 16, pp. 777–784, 1999.

[38] Habashi, W., Dompierre, J., Bourgault, Y., Yahia, A., Fortin, M., and Vallet, M., "Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part I: general principles," *International Journal for Numerical Methods in Fluids*, vol. 32, pp. 725–744, 2000.

[39] Hirsch, C., *Numerical Computation of Internal and External Flows Volume 1:Fundamentals of Numerical Discretization*. John Wiley and Sons, 1997.

[40] Houston, P. and Sli, E., "Hp-adaptive discontinuous galerkin finite element methods for first-order hyperbolic problems," *Journal on Scientific Computing*, vol. 23, no. 4, pp. 1226–1252, 2001.

[41] Ingber, L., "Simulated annealing: Practice versus theory," *Journal of Mathematical and Computer Modelling*, vol. 18, no. 11, 1993.

[42] Jansen, K., Shephard, M., and Beall, M., "On anisotropic mesh generation and quality control in complex flow problems," in *10th International Meshing Roundtable*, (Sandia National Laboratories), 2001.

[43] Johnson, D., Aragon, C., McGeoch, L., and Schevon, C., "Optimization by simulated annealing: An experimental evaluation; Part I, graph partitioning," *Operations Research*, vol. 6, pp. 865–896, 1989.

[44] Kirkpatrick, S., Gelatt, C., and Jr., M. V., "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[45] Knupp, P., "Algebraic mesh quality metrics," *SIAM Journal on Scientific Computing*, vol. 23, no. 1, pp. 193–218, 2001.

[46] Knupp, P., "Algebraic mesh quality metrics for unstructured initial meshes," *Finite Elements in Analysis and Design*, vol. 39, pp. 217–241, 2003.

[47] Krumbein, A., "Transitional flow modeling and application to high lift multi-element airfoil configurations," *Journal of Aircraft*, vol. 40, pp. 786–794, 2003.

[48] Lewis, R., Zheng, Y., and Gethin, D., "Three dimensional unstructured mesh generation: Part 3. volume meshes," *Computer methods in applied mechanics and engineering*, vol. 134, pp. 285–310, 1995.

[49] Lin, P., Baker, T., Martinelli, L., and Jameson, A., "Two-dimensional implicit time-dependent calculations on adaptive unstructured meshes with time evolving boundaries," *International Journal for Numerical Methods in Fluids*, vol. 50, pp. 199–218, 2006.

[50] Liu, A. and Joe, B., "On the shape of a tetrahedra based from bisection," *Mathematics of Computations*, vol. 63, pp. 141–154, 1994.

[51] Lo, S., "3D anisotropic mesh refinement in compliance with a general metric specification," *Finite Elements in Analysis and Design*, vol. 38, pp. 3–19, 2001.

[52] Lohner, J., "Adaptive H-refinement on 3D unstructured grids for transient problems," *International Journal for Numerical Methods in Engineering*, vol. 14, pp. 1407–1419, 1992.

[53] Machiels, L., "A posteriori finite elements bounds for output functionals of discontinuous Galerkin discretizations of parabolic problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 3401–3411, 2001.

[54] Marcum, D. and Weatherill, N., "Aerospace applications of solution adaptive finite element analysis," *Computer Aided Geometric Design*, vol. 12, pp. 709–731, 1995.

[55] Miller, K., "Moving finite elements II," *SIAM Journal on Numerical Analysis*, vol. 18, p. 1003, 1981.

[56] Mohammadi, B., *Fluid Dynamics Computation with NSC2KE A User-Guide Release 1.0*. INRIA, May 1994.

[57] Paige, C. and Saunders, M., "Solution of sparse indefinite systems of linear equations," *SIAM J. Numer. Anal.*, vol. 12, no. 4, pp. 617–629, 1975.

[58] Pain, C., Umpleby, A., Oliveria, C. D., and Goddard, A., "Tetrahedral mesh optimisation and adaptation for steady state and transient finite element calculations," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 3771–3796, 2001.

[59] Peraire, J. and Patera, A., "Asymptotic a posteriori finite element bounds for the outputs of noncoercive problems: Helmholtz and Burgers equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 171, pp. 77–86, 1999.

[60] Peraire, J., Vahdati, M., Morgan, K., and Zienkiewicz, O., "Adaptive remeshing for compressible flow computations," *Journal of Computational Physics*, vol. 72, pp. 449–466, 1987.

[61] Pirzadeh, S. H., "An adaptive unstructured grid method by grid subdivision, local remeshing and grid movement," Tech. Rep. AIAA 99-3255, 14th AIAA Computational Fluid Dynamics Conference, Virginia, 1999.

[62] Popiolek, T. and Awruch, A., "Numerical simulation of incompressible flows using adaptive unstructured meshes and pseudo-compressibility hyphothesis," *Advances in Engineering Software*, vol. 37, pp. 260–274, 2006.

[63] Potapczuk, M. and Berkowitz, B., "An experimental investigation of multi-element airfoil ice-accretion and resulting performance degradation," Tech. Rep. NASA TM-101441, NASA, Cleveland, Ohio, 1989.

[64] Prudhomme, S. and Oden, J., "A posteriori error estimation and error control for finite element approximations of the time-dependent Navier-Stokes equations," *Computer Aided Geometric Design*, vol. 33, pp. 247–262, 1999.

[65] Rannacher, R., "Adaptive Galerkin finite element methods for partial differential equations," *International Journal for Numerical Methods in Engineering*, vol. 128, pp. 205–233, 2001.

[66] Rardin, R. R., *Optimization in operations research*. New Jersey: Prentice Hall, 1998.

[67] Remaki, L., Lepage, C., and Habashi, W., "Efficient anisotropic mesh adaptation on weak and multiple shocks," in *42nd AIAA Aerospace Sciences Meeting and Exhibit*, (Reno, NV), January 2004.

[68] Rivara, M. and Inostroza, P., "Using longest-side bisection techniques for the automatic refinement of Delaunay triangulations," *International Journal for Numerical Methods in Engineering*, vol. 63, pp. 141–154, 1994.

[69] Salamon, P., Sibani, P., and Frost, R., *Facts, Conjectures, and Improvements for Simulated Annealing*. Philadelphia: SIAM Monographs on Mathematical Modeling and Computation, 2002.

[70] Saltel, E. and Hecht, F., *EMC2 Wysiwyg 2D finite element mesh generator A User-Guide*. INRIA, 1995.

[71] Scalabrin, L. and J.L.F.Azevedo, "Adaptive mesh refinement and coarsening for aerodynamic flow simulations," *International Journal for Numerical Methods in Engineering*, vol. 45, pp. 1107–1122, 2004.

[72] Schumaker, L., "Computing optimal triangulations using simulated annealing," *Computer-Aided Design*, vol. 10, pp. 329–345, 1993.

[73] Semper, B. and Liao, G., "A moving grid finite-element method using grid deformation," *Numerical Methods for PDE's*, vol. 11, pp. 603–615, 1995.

[74] Shabana, A., *Computational Dynamics*. John Wiley and Sons, 2001.

[75] Shephard, M. and Georges, M., "Automatic three-dimensional mesh generation by the finite octree technique," *International Journal for Numerical Methods in Engineering*, vol. 32, pp. 709–749, 1991.

[76] Shephard, M. and Georges, M., "Reliability of automatic 3D mesh generation," *Computer Methods in Applied Mechanics and Engineering*, vol. 101, pp. 443–462, 1992.

[77] Shewchuk, R., "An introduction to conjugate gradient method without the agonizing pain," tech. rep., School of Computer Science, Carnegie Mellon University, 1994.

[78] Slawig, T., "Domain optimization of a multi-element airfoil using automatic differentiation," *Advances in Engineering Software*, vol. 32, pp. 225–237, 2001.

[79] Tam, A., Yahia, A., Robichaud, M., Moore, M., Kozel, V., and Habashi, W., "Anisotropic mesh adaption for 3D flows on structured and unstructured grids," *Computer Methods in Applied Mechanics and Engineering*, vol. 189, pp. 1205–1230, 2000.

[80] Tautges, T., "The generation of hexahedral meshes for assembly geometry: Survey and progress," *International Journal for Numerical Methods in Engineering*, vol. 50, pp. 2617–2642, 2001.

[81] Tchon, K., Dompierre, J., and Camerero, R., "Automated refinement of conformal quadrilateral and hexahedral meshes," *International Journal for Numerical Methods in Engineering*, vol. 59, pp. 1539–1562, 2004.

[82] Tchon, K., Khachan, M., Guibault, F., and Camerero, R., "Three-dimensional anisotropic geometric metrics based on local domain curvature and thickness," *Computer-Aided Design*, vol. 37, pp. 173–187, 2005.

[83] Tezduyar, T. E., Behr, M., and Liou, J., "A new strategy for finite element computations involving moving boundaries and interfaces -the deforming-spatial-domain/space-time procedure: I. the concept of preliminary tests," *Computer Methods in Applied Mechanics and Engineering*, vol. 94, pp. 339–351, 1992.

[84] Vallet, M., *Génération de Maillages Éléments Finis Anisotropes et Adaptatifs*. PhD dissertation, Université Pierre et Marie Curie, Paris VI, France, 1992.

[85] Venditti, A., *Grid Adaptation for Functional Outputs of Compressible Flow Simulations*. PhD dissertation, Massachusetts Institute of Technology, Massachusetts, USA, 2002.

[86] Venkatakrishnan, V. and Mavripilis, D., "Implicit method for the computation of unsteady flows on unstructured grids," *Journal of Computational Fluids*, vol. 127, pp. 380–397, 1996.

[87] Walter, M., Abdu, A., Silva, L. D., and J.L.F.Azevedo, "Evaluation of adaptive mesh refinement and coarsening for the computation of compressible flows on unstructured meshes," *International Journal for Numerical Methods in Engineering*, vol. 49, pp. 999–1014, 2005.

[88] Webster, B., Shephard, M., Rusak, Z., and Flaherty, J., "Automated adaptive time-discontinuous finite element method for unsteady compressible airfoil aerodynamics," *AIAA Journal*, vol. 32, no. 4, pp. 748–757, 1994.

[89] Yahia, A., Baruzzi, G., Habashi, W., Fortin, M., Dompierre, J., and Vallet, M., "Anisotropic mesh adaptation: towards user-independent, mesh-independent and solver-independent CFD. Part II: structured grids," *International Journal for Numerical Methods in Fluids*, vol. 39, pp. 657–673, 2002.

[90] Yannis, K. and Kavouklis, C., "A dynamic adaptation scheme for general 3-d hybrid meshes," *Computer Methods in Applied Mechanics and Engineering*, vol. 194, pp. 5019–5050, 2005.

[91] Zegeling, P., "R-refinement for evolutinary PDE's with finite elements or finite differences," *Applied Numerical Mathematics*, vol. 26, pp. 97–104, 1998.

[92] Zegeling, P., "On resistive mhd models with adaptive moving meshes," *Journal of Scientific Computing*, vol. 24, pp. 263–284, 2005.

[93] Zienkiewicz, O., Boroomand, B., and Zhu, J., "Recovery procedures in error estimation and adaptivity. Part I: Adaptivity in linear problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 176, pp. 111–125, July 1999.

VITA

Nazmiye Acikgoz was born on June 16th, 1976 in Ankara, Turkey. In 1998, she received her B.S degree from prestigious Middle East Technical University (Ankara), in Aeronautical Engineering. She joined School of Aerospace Engineering at Georgia Institute of Technology in 2001 for her graduate studies where she completed her M.S and Ph.D in 2003 and 2007 respectively. Her areas of research interest include computational fluid dynamics, adaptive meshing, numerical methods, aerodynamics and gas dynamics.